

Analyse spectrale paramétrique

Majeure SIR – Systèmes Interactifs et Robotiques

Stéphane Rossignol – stephane.rossignol@supelec.fr

2013 – 2014

Les slides sont disponibles ici

[http://www.metz.supelec.fr/metz/personnel/
rossignol/slides_analyse.pdf](http://www.metz.supelec.fr/metz/personnel/rossignol/slides_analyse.pdf)

Version susceptible d'évolution

Introduction

Modèles autorégressifs

Pisarenko

MUSIC

Prony

Localisation de sources

Performances, choix de l'ordre, modèles MA et ARMA

Plus de performances

Sélection de l'ordre

Modèles MA

Modèles ARMA

Plan

- ▶ Introduction

Introduction

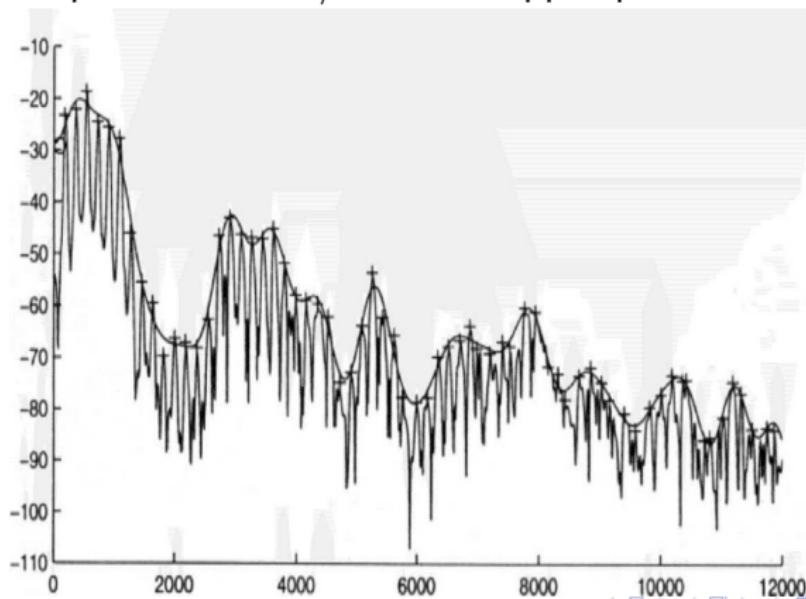
- ▶ dans de nombreuses applications, on cherche à récupérer de l'information à partir de N observations $\{x(1) \dots x(N)\}$ temporelles (ou spatiales)
- ▶ bien que les domaines du temps (ou de l'espace) et de la fréquence soient duals, l'information est souvent plus intelligible dans le domaine des fréquences
- ▶ d'où la nécessité de développer des outils d'analyse spectrale d'un signal
- ▶ l'analyse spectrale sert souvent de pré-traitement en vue de la classification, de la reconnaissance, de la compression, du filtrage, de la détection, etc., des signaux

Introduction

- ▶ on suppose ici (méthodes paramétriques) que le signal appartient à une famille de signaux qui dépend d'un ensemble de paramètres Θ
 - ▶ il s'agit dans un premier temps d'estimer ces paramètres Θ
 - ▶ puis, une estimation du spectre peut être obtenue (par dualité)
- ▶ ces méthodes sont aussi dites à haute-résolution, parce qu'elles sont capables de dépasser la résolution fréquentielle des méthodes non-paramétriques (particulièrement pour N petit)
- ▶ elles permettent aussi d'estimer les enveloppes spectrales

Introduction

On veut attraper les raies et/ou l'enveloppe spectrale :



Introduction

- ▶ le problème fondamental de l'analyse spectrale, c'est le principe d'incertitude de Gabor (c'est l'équivalent pour le traitement du signal du principe d'incertitude d'Heisenberg)
- ▶ les signaux analysés ne sont stationnaires/pseudo-stationnaires que sur de très courts intervalles de temps (de l'ordre de $\Delta T = 40 \text{ ms}$ pour le son)
- ▶ de ce fait, on ne peut analyser efficacement que des trames de signal de cet ordre de taille ΔT
- ▶ du coup, on limite énormément la résolution fréquentielle Δf qu'on peut espérer

Introduction

- ▶ c'est-à-dire, si l'on admet que le signal est composé de raies (fréquences f_i , amplitudes a_i , phases ϕ_i), qu'on ne peut pas espérer séparer complètement des raies trop proches en f
- ▶ en effet, si on appelle Δf la distance en fréquence entre deux raies, Gabor dit que $\Delta T \Delta f = \text{constante}$
- ▶ on peut dépasser cette limite, par exemple avec les méthodes haute-résolution qu'on va voir, mais il y a un **coût**
- ▶ il est porté sur l'estimation des amplitudes a_i et des phases ϕ_i , notamment
- ▶ ainsi que sur la complexité des procédures d'estimation (valeurs de divers ordres, contrôlant le nombre de paramètres)

Bibliographie

- ▶ Steven M. Kay et Stanley Lawrence Marple Jr. "Spectrum analysis – a modern perspective", Proceedings of the IEEE, Volume 69, pages 1380-1419, novembre 1981
- ▶ Gilles Fleury. Analyse spectrale – Méthodes non-paramétriques et paramétriques, Ellipses, 2001
- ▶ J. E. Rooijackers, "Algorithms for Speech Coding Systems Based on Linear Prediction", EUT Report 92-E-260, 1992
 - ▶ le point de vue d'un practitioner
 - ▶ disponible sur le web

- ▶ Introduction
- ▶ Modèles AR

Modèles ARMA en général

Un certain nombre de processus déterministes ou aléatoires, rencontrés dans le monde réel, peuvent être modélisés de cette façon :

$$x_n = \sum_{l=0}^q b_l n_{n-l} - \sum_{k=1}^p a_k x_{n-k}$$

où x_n est le signal à modéliser, a_k et b_l les paramètres à déterminer, et n_n un bruit blanc gaussien

Modèles AR – AR=Autorégressifs

- ▶ dans ce cas : $q = 0$ et $b_0 = 1$
- ▶ le modèle est alors : $x_n = -\sum_{k=1}^p a_k x_{n-k} + n_n$ M1
- ▶ il faut estimer les paramètres a_k , en minimisant la variance σ^2 du bruit n
- ▶ plusieurs approches sont possibles (un grand nombre d'approches sont possibles)

Algorithmes

- ▶ donc, maintenant, il faut résoudre le problème, et ce, ou bien à partir directement du signal x ; ou bien à partir des coefficients d'autocorrélation \hat{R}_{xx} estimés, donc imparfaits
 - ▶ ils sont estimés et dans une certaine mesure mal estimés parce que l'horizon d'observation est limité à N échantillons
- ▶ les trois algorithmes qu'on va examiner un peu en détails sont :
 - ▶ celui de **Levinson-Durbin** (certains l'appellent simplement de Levinson; d'autres – même – de Yule-Walker)
 - ▶ celui de **Burg**
 - ▶ celui de **Marple**

Algorithme 1 – Levinson-Durbin

Page intentionnellement laissée blanche

Algorithme 1 – Levinson-Durbin

- ▶ on estime d'abord les coefficients d'autocorrélation \hat{R}_{xx}
- ▶ à partir de ces coefficients d'autocorrélation, il est possible d'obtenir les coefficients AR
- ▶ l'étape préliminaire consiste à obtenir les équations de Yule-Walker, qui lient les paramètres AR aux coefficients d'autocorrélation de x , R_{xx}

Les équations de Yule-Walker

- ▶ on a :

$$R_{xx}(k) = E[x_{n+k}x_n^*]$$

$$= E \left[x_n^* \left(- \sum_{l=1}^p a_l x_{n-l+k} + n_{n+k} \right) \right]$$

$$= - \sum_{l=1}^p a_l R_{xx}(-l+k) + E[n_{n+k}x_n^*]$$

- ▶ or, la relation **M1** entre n_n et x_n peut aussi être considérée comme une opération de filtrage, avec n_n l'entrée et x_n la sortie

Les équations de Yule-Walker

- ▶ la fonction de transfert de ce filtre stable, linéaire et causal, est :

$$H(z) = \frac{1}{A(z)} = \frac{1}{\sum_{m=0}^p a_m z^{-m}} \text{ avec } a_0 = 1$$

- ▶ on appelle h_l la réponse impulsionnelle de ce filtre
- ▶ comme le système est causal, $h_l = 0 \forall l < 0$
- ▶ on a alors :

$$E[n_{n+k}x_n^*] = E \left[n_{n+k} \sum_{l=0}^{+\infty} h_l^* n_{n-l}^* \right]$$

Les équations de Yule-Walker

- ▶ ceci devient :

$$E[n_{n+k}x_n^*] = \sum_{l=0}^{+\infty} h_l^* E[n_{n+k} n_{n-l}^*]$$

- ▶ or n_n est un bruit blanc ; sa variance est σ^2 (elle devra être estimée elle aussi) et bien sûr son autocorrélation est :

$$E[n_{n+k} n_{n-l}^*] = \sigma^2 \delta_{k+l}$$

- ▶ donc :

$$\begin{aligned} E[n_{n+k}x_n^*] &= \sigma^2 h_{-k}^* \\ &= 0 \text{ pour } k > 0 \text{ et} \\ &\quad \sigma^2 h_0^* \text{ pour } k = 0 \end{aligned}$$

Les équations de Yule-Walker

- ▶ or : $h_0 = \lim_{z \rightarrow +\infty} H(z) = 1$ (théorème de la valeur initiale)
- ▶ du coup, on a (c'est l'une des formes des équations de Yule-Walker) :

$$R_{xx}(k) = \begin{cases} - \sum_{l=1}^p a_l R_{xx}(k-l) & \text{pour } k > 0 \\ - \sum_{l=1}^p a_l R_{xx}(k-l) + \sigma^2 & \text{pour } k = 0 \end{cases}$$

Les équations de Yule-Walker

- ▶ pour déterminer les paramètres AR, il suffit de choisir p équations parmi l'ensemble ci-dessus
- ▶ l'ensemble d'équations qui requiert les plus petits décalages k est celui qu'on obtient en prenant $k = 1, 2 \dots p$
 - ▶ ça correspond à ceux qui sont le mieux estimés !
- ▶ pour estimer σ^2 , il faut une équation supplémentaire
- ▶ il faut aussi se souvenir que $R_{xx}(-m) = R_{xx}^*(m)$

Les équations de Yule-Walker

- ▶ du coup, sous forme matricielle, ça s'écrit :

$$\begin{bmatrix} R_{xx}(0) & R_{xx}(-1) & \dots & R_{xx}(-(p-1)) \\ R_{xx}(1) & R_{xx}(0) & \dots & R_{xx}(-(p-2)) \\ \vdots & \vdots & \ddots & \vdots \\ R_{xx}(p-1) & R_{xx}(p-2) & \dots & R_{xx}(0) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix} = - \begin{bmatrix} R_{xx}(1) \\ R_{xx}(2) \\ \vdots \\ R_{xx}(p) \end{bmatrix}$$

- ▶ ou encore si on incorpore l'estimation de σ^2 :

$$\begin{bmatrix} R_{xx}(0) & R_{xx}(-1) & \dots & R_{xx}(-p) \\ R_{xx}(1) & R_{xx}(0) & \dots & R_{xx}(-(p-1)) \\ \vdots & \vdots & \ddots & \vdots \\ R_{xx}(p) & R_{xx}(p-1) & \dots & R_{xx}(0) \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \\ \vdots \\ a_p \end{bmatrix} = \begin{bmatrix} \sigma^2 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Algorithme 1 – Levinson-Durbin

- ▶ l'algorithme de Levinson-Durbin est un algorithme d'inversion de la matrice carrée ci-dessus
 - ▶ il requiert $o(p^2)$ opérations $\Rightarrow o(p^3)$ pour le pivot de Gauss
- ▶ il n'est pas seulement efficace : il révèle aussi des propriétés intéressantes des processus AR
- ▶ il est itératif : on commence avec un ordre p égal à 1 et on va jusqu'à l'ordre p désiré
 - ▶ l'ordre p auquel il faut s'arrêter n'est lui non plus pas connu
 - ▶ on va voir plus loin des techniques pour décider à quel ordre p il faut s'arrêter
 - ▶ on obtient les ensembles de paramètres : $\{a_{1,1}, \sigma_1^2\}$,
 $\{a_{2,1}, a_{2,2}, \sigma_2^2\}$, ..., $\{a_{p,1}, a_{p,2}, \dots, a_{p,p}, \sigma_p^2\}$
 - ▶ notez l'indice supplémentaire, qui indique l'ordre p atteint

Algorithme 1 – Levinson-Durbin

- ▶ à l'itération 1 ($p = 1$), on a :

$$a_{1,1} = -\frac{\hat{R}_{xx}(1)}{\hat{R}_{xx}(0)} \quad \text{E1}$$

$$\sigma^2 = (1 - |a_{1,1}|^2) \hat{R}_{xx}(0) \quad \text{E2}$$

- ▶ qui s'obtient facilement à partir de :

$$\begin{bmatrix} \hat{R}_{xx}(0) & \hat{R}_{xx}^*(1) \\ \hat{R}_{xx}(1) & \hat{R}_{xx}(0) \end{bmatrix} \begin{bmatrix} 1 \\ a_{1,1} \end{bmatrix} = \begin{bmatrix} \sigma^2 \\ 0 \end{bmatrix}$$

- ▶ de la ligne 2 on tire l'estimation de $a_{1,1}$, et aussi $\hat{R}_{xx}(1) = -a_{1,1}\hat{R}_{xx}(0)$ et donc $\hat{R}_{xx}^*(1) = -a_{1,1}^*\hat{R}_{xx}(0)$ (car $\hat{R}_{xx}(0)$ est réel)
- ▶ de la ligne 1 on tire $\sigma^2 = \hat{R}_{xx}(0) + a_{1,1}\hat{R}_{xx}^*(1)$ etc.

Algorithme 1 – Levinson-Durbin

- ▶ aux itérations $k = 2, 3\dots$ on a :

$$a_{k,k} = -\frac{\hat{R}_{xx}(k) + \sum_{l=1}^{k-1} a_{k-1,l} \hat{R}_{xx}(k-l)}{\sigma_{k-1}^2} \quad \text{E3}$$

$$a_{k,i} = a_{k-1,i} + a_{k,k} a_{k-1,k-i}^* \text{ pour } i < k \quad \text{E4}$$

$$\sigma_k^2 = (1 - |a_{k,k}|^2) \sigma_{k-1}^2 \quad \text{E5}$$

- ▶ note : c'est la récursion de Levinson-Durbin

Calculs compagnons – 1. Estimation de l'autocorrélation

- ▶ estimateur non-biaisé (jusqu'à $m \leq N - 1$) :

$$\hat{R}_{xx}(m) = \frac{1}{N-m} \sum_{n=0}^{N-m-1} x_{n+m} x_n^*$$

- ▶ estimateur biaisé (jusqu'à $m \leq N - 1$) :

$$\hat{R}'_{xx}(m) = \frac{1}{N} \sum_{n=0}^{N-m-1} x_{n+m} x_n^* \text{ E6}$$

- ▶ la deuxième forme est plutôt utilisée, car pour beaucoup de données elle tend à présenter une erreur quadratique plus petite

Calculs compagnons – 2. Estimation de la DSP

- ▶ n est un bruit blanc, donc sa DSP est égale à σ^2/f_e
- ▶ la formule des interférences dit que la DSP de x par rapport à celle de n est :

$$P_{xx}(z) = H(z)H^*(1/z^*)P_n(z) = \frac{1}{A(z)A^*(1/z^*)}P_n(z)$$

- ▶ elle est estimée pour $z = \exp(j2\pi f/f_e)$ pour $-f_e/2 \leq f \leq f_e/2$
- ▶ du coup, on a :

$$P_{xx}(f) = |H(f)|^2 P_n(f) = \frac{\sigma^2}{f_e \left| 1 + \sum_{k=1}^p a_k \exp(-j2\pi kf/f_e) \right|^2} \quad E7$$

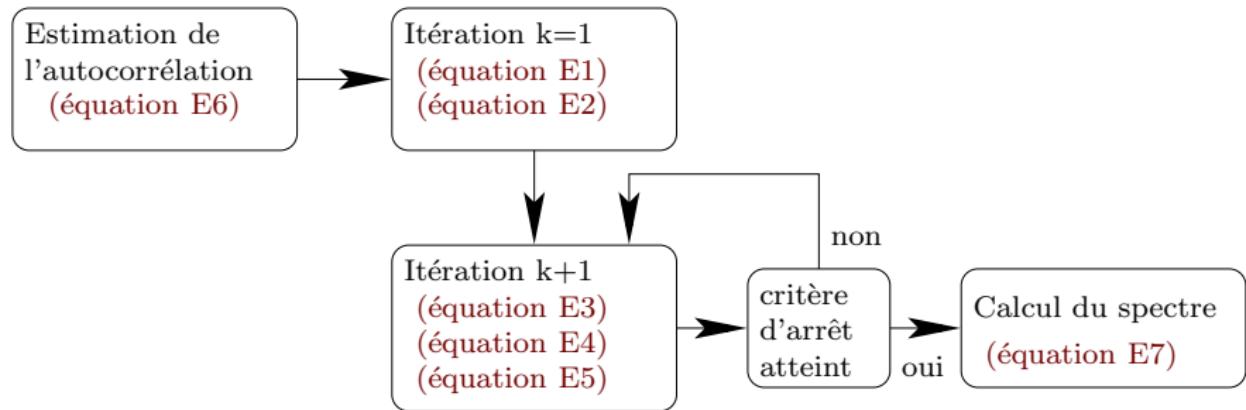
Algorithme 1 – Levinson-Durbin – Remarques

- ▶ l'algorithme fournit les paramètres AR pour tous les modèles d'ordres inférieurs à p et qui collent au mieux aux données
- ▶ on pourrait utiliser ça comme critère d'arrêt :
 - ▶ σ_k^2 décroissant avec k (car $|a_{k,k}| \leq 1$), quand σ_k^2 est suffisamment petit, on stoppe l'algorithme
 - ▶ de plus, si le processus est réellement AR d'ordre p , alors on a $a_{k,k} = 0$ et $\sigma_k^2 = \sigma^2$ pour $k > p$
 - ▶ malheureusement, ça ne se passe pas comme ça dans la réalité, comme on va le voir

Algorithme 1 – Levinson-Durbin – Remarques

- ▶ les paramètres $\{a_{1,1}, a_{2,2}, \dots, a_{p,p}\}$ sont appelés les coefficients de réflexion
 - ▶ on les note aussi $\{k_1, \dots, k_p\}$
 - ▶ si toutes les $|k_i|$ sont ≤ 1 , les coefficients d'autocorrélation \hat{R}_{xx} utilisés sont valides
 - ▶ alors, tous les pôles de $A(z)$ sont dans le cercle unité (ou sur lui)
 - ▶ si on a $|k_k| = 1$ pour un certain k , alors le signal est purement harmonique (composé de sinusoïdes)

Algorithme 1 – Levinson-Durbin – Résumé



Algorithme 1 – Levinson-Durbin – Code matlab

```
function [aa, sigma2, ref, ff, mydsp] = mylevinsondurbin (xx, pp, fe)

acf = xcorr(xx, pp+1, 'biased');
acf(1 :pp+1) = [] ; %% on enlève la partie négative
acf(1) = real(acf(1)); %% Levinson-Durbin requiert c(1)==conj(c(1))

ref = zeros(pp,1);
gg = -acf(2)/acf(1);
aa = [ gg ];
sigma2 = real( ( 1 - gg*conj(gg)) * acf(1) ); %% real : enlève une éventuelle partie imaginaire résiduelle
ref(1) = gg;
for tt = 2 : pp
    gg = -(acf(tt+1) + aa * acf(tt :-1 :2)) / sigma2;
    aa = [ aa + gg*conj(aa(tt-1 :-1 :1)), gg ];
    sigma2 = sigma2 * ( 1 - real(gg*conj(gg)) );
    ref(tt) = gg;
end;
aa = [1, aa];

interm2=-j*2*pi/fe*[1 :pp];
ff=-fe/2 :10 :fe/2;
for ii=1 :length(ff)
    interm=1.+aa(2 :end)*exp(interm2*ff(ii))';
    mydsp(ii) = sigma2./((interm.*conj(interm))); %%% densité spectrale de puissance
end;
```

Algorithme 1 – Levinson-Durbin – Premiers tests

On va faire quelques tests, avec des signaux (sons) simples pour vérifier qu'on obtient bien des densités spectrales de puissance (DSP ou PSD) intéressantes. Les 4 tests qu'on considère ici sont :

- ▶ on a une sinusoïde pure \Rightarrow est-ce qu'on arrive à la mettre en évidence ?
- ▶ on a deux sinusoïdes proches en fréquence \Rightarrow est-ce qu'on arrive à les discriminer ?
- ▶ on a un signal compliqué : une somme de sinusoïdes harmoniques \Rightarrow est-ce qu'on arrive à obtenir une enveloppe spectrale, modélisant ce signal, et/ou les n partiels ?
- ▶ on a un signal compliqué : un bruit rose \Rightarrow est-ce qu'on arrive à obtenir une enveloppe spectrale, modélisant ce signal ?

Algorithme 1 – Levinson-Durbin – Et ça donne quoi ?

mylevinsondurbin.m

Paramètres de l'exemple :

- ▶ $f_e = 32000$ Hz
- ▶ signal : une sinusoïde présente
 - ▶ de fréquence 440 Hz
 - ▶ et de phase aléatoire, tirée entre 0 et 2π
 - ▶ de taille 1280 échantillons (40 ms)
- ▶ $p = 4$

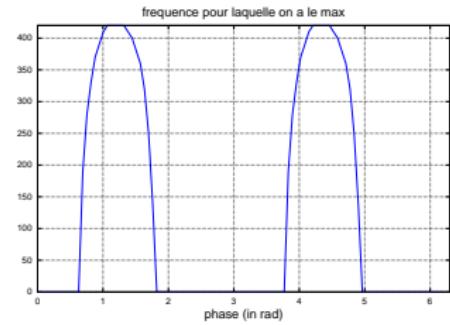
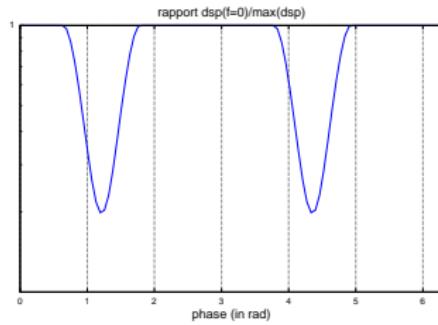
Algorithme 1 – Levinson-Durbin – Et ça donne quoi ?

scriptphaselevinson.m

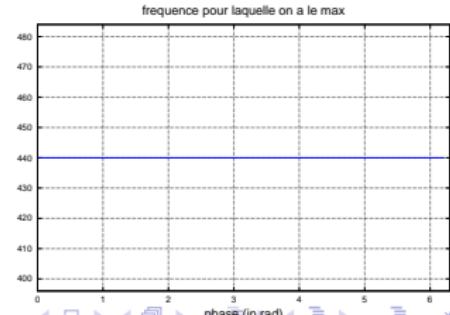
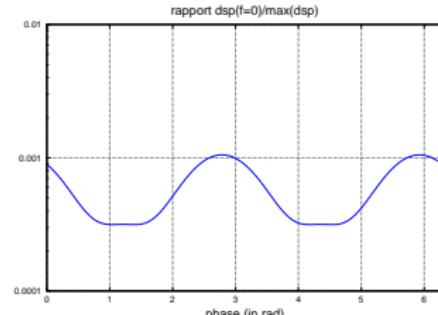
- ▶ on regarde ce qui se passe en fonction de la phase de la sinusoïde
- ▶ on mesure le rapport entre l'amplitude de la DSP en $f = 0$ et l'amplitude maximum de la DSP (sensée se trouver à $\pm 440 \text{ Hz}$)
- ▶ on regarde où se trouve en réalité l'amplitude maximum de la DSP
- ▶ ce pour différents ordres : $p = 2, p = 3, p = 4, \dots$

Algorithme 1 – Levinson-Durbin – Et ça donne quoi ?

$p = 2 :$



$p = 20 :$



Algorithme 1 – Levinson-Durbin – Et ça donne quoi ?

Dans le tableau suivant, on donne le nombre de fois (sur 100) où la fréquence du maximum a été trouvée entre 400 Hz et 480 Hz.

p	2	3	4	20	30	50	100	150
%	16	50	100	100	100	100	100	100

Un ordre trop élevé ne semble jamais devenir problématique.

Algorithme 1 – Levinson-Durbin – Intérêt

- ▶ **on n'a pas encore vu pourquoi les méthodes haute-résolution présentent un intérêt par rapport à disons la FFT !**
- ▶ il faudrait inspecter ce qu'on obtient quand on a un signal où deux sinusoïdes de fréquences proches sont présentes
- ▶ du coup, on simule un signal composé de deux sinusoïdes séparées de 26 Hz (440 Hz et 466 Hz) et de même amplitude
- ▶ et on essaie de les discriminer, soit en utilisant la FFT – fenêtrage de Blackman, soit en utilisant la méthode AR de Levinson-Durbin
- ▶ la longueur du signal reste 1280 échantillons (une seule trame de 40 ms)

Algorithme 1 – Levinson-Durbin – Intérêt

- ▶ la DSP FFT et la DSP AR sont calculées sur 32768 points
- ▶ pour chaque DSP, on devrait donc avoir deux pics dans les fréquences positives (on ne s'intéresse qu'à elles), qu'il faut détecter
- ▶ cette détection est en soit un problème !
- ▶ il ne faudrait pas que ce problème vînt perturber nos mesures de performances !
- ▶ on regarde entre 400 Hz et 506 Hz (une inspection visuelle montre qu'en dehors de cette bande de fréquences, les DSP ne présentent pas de pics significatifs)

Algorithme 1 – Levinson-Durbin – Intérêt

- ▶ la méthode de détection utilisée est simple : on détecte les maximums locaux et les minimums locaux dans cette bande
 - ▶ le code dit pour les maximums locaux :

```
if ( (dsp(ii-1)<=dsp(ii) & dsp(ii)>dsp(ii+1)) |  
      (dsp(ii-1)<dsp(ii) & dsp(ii)>=dsp(ii+1)) )
```

- ▶ il y a un maximum local
- ▶ ce qui n'assure pas qu'il y a 1 minimum local entre 2 maximums locaux
- ▶ notez aussi qu'on n'inspecte pas précisément les positions fréquentielles des pics
- ▶ puis on vérifie qu'il y a bien 2 maximums locaux, 1 minimum local, et que celui-ci se situe entre les 2 maximums locaux

Algorithme 1 – Levinson-Durbin – Intérêt

- ▶ la méthode est certes imparfaite, mais elle est exactement la même pour les deux méthodes d'extraction de la DSP, donc on peut s'attendre à ce qu'elle n'en favorise aucune
- ▶ le résultat obtenu dépend de la phase respective des 2 sinus
- ▶ il dépend aussi de l'ordre du modèle AR (qu'il faut prendre grand) et de la méthode de fenêtrage pour la FFT

Algorithme 1 – Levinson-Durbin – Intérêt

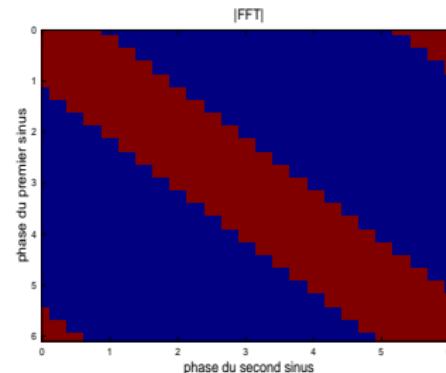
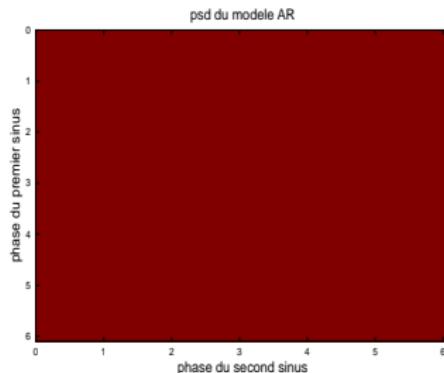
► voir :

scriptdoublonlevinson.m

Algorithme 1 – Levinson-Durbin – Intérêt

Levinson-Durbin (ordre : 300) : 100 % de succès

FFT (fenêtre : Blackman) : 32 % de succès



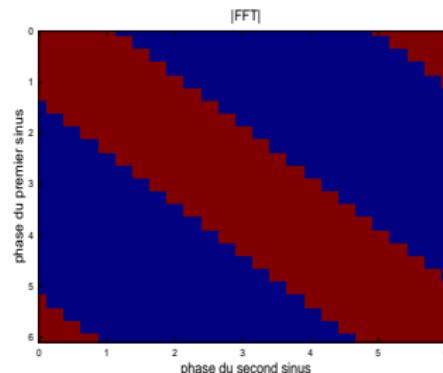
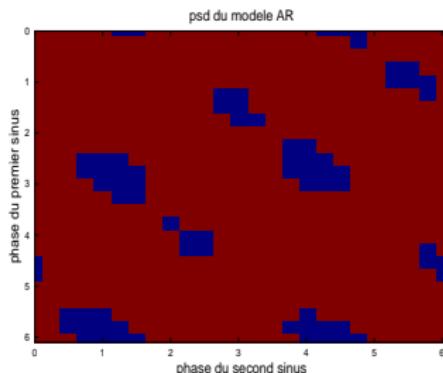
En rouge : les 2 sinusoïdes ont été séparées

En bleu : les 2 sinusoïdes n'ont pas été séparées

Algorithme 1 – Levinson-Durbin – Intérêt

Levinson-Durbin (ordre : 265) : 88.32 % de succès

FFT (fenêtre : Hanning) : 40 % de succès



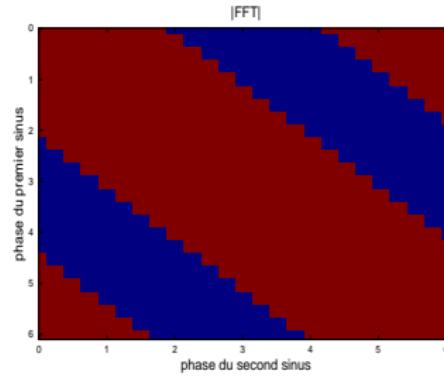
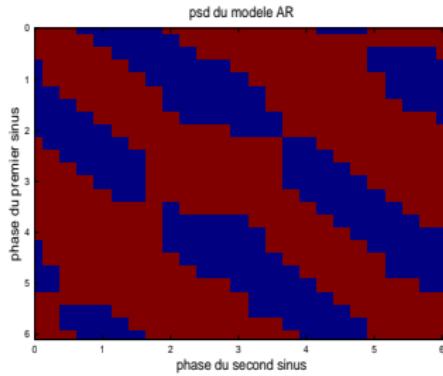
En rouge : les 2 sinusoïdes ont été séparées

En bleu : les 2 sinusoïdes n'ont pas été séparées

Algorithme 1 – Levinson-Durbin – Intérêt

Levinson-Durbin (ordre : 250) : 63.36 % de succès

FFT (Rectangulaire) : 64 % de succès **Attention : les lobes secondaires sont très gênants ; il faut réduire de 38 % la zone d'exploration ([420 486 Hz]), sinon on a des pics et des creux parasites, c'est-à-dire renforcer la connaissance a priori qu'on a du signal**



Algorithme 1 – Levinson-Durbin – Intérêt

- ▶ ça marche pour de grands ordres p (alors que $p = 4$ est théoriquement l'ordre minimum qui permet de détecter 2 sinusoïdes, notre cas ici !)
- ▶ on atteint 100 % de bonnes détections
- ▶ on fait mieux qu'avec la FFT, pour laquelle, en tirant sur la corde, on obtient au mieux 64 % de bonnes détections
- ▶ mise en place de méthodes d'estimation de l'ordre p à utiliser
- ▶ la plage d'ordres p pour lesquels la méthode marche bien est à étudier plus

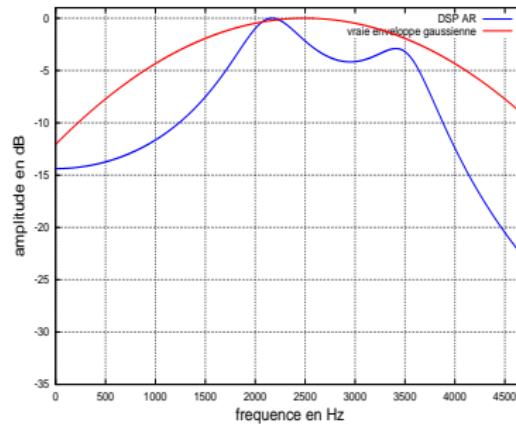
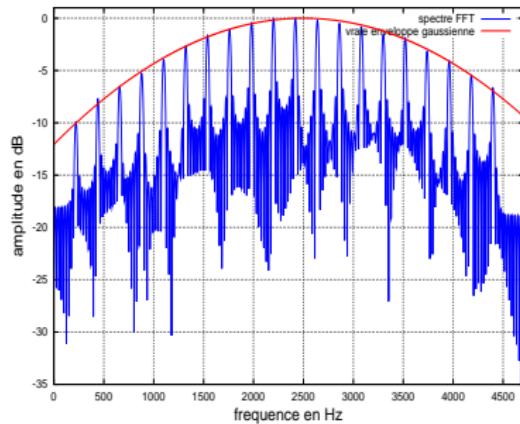
Algorithme 1 – Levinson-Durbin – Enveloppe

On a un signal harmonique (composé d'une somme de partiels harmoniques). Le signal considéré :

- ▶ fréquence fondamentale : $f_0 = 220 \text{ Hz}$
- ▶ nombre de partiels : 20
- ▶ amplitudes des partiels : elles suivent une gaussienne centrée en $f_{max} = 2500 \text{ Hz}$ et d'écart-type 1500
 - ▶ voir transparent suivant
- ▶ phases des partiels : tirées aléatoirement entre 0 et 2π
- ▶ la longueur du signal est toujours 1280 échantillons, et la fréquence d'échantillonnage 32000 Hz

Algorithme 1 – Levinson-Durbin – Enveloppe

Exemple de spectre FFT et d'enveloppe spectrale AR ($p = 10$, norme 1) obtenus avec le signal précédent :



Algorithme 1 – Levinson-Durbin – Enveloppe

Note : bien sûr, les pics de la FFT collent bien à la vraie enveloppe, mais par contre il faut encore détecter automatiquement ces pics, pour obtenir l'enveloppe spectrale !

Mesures :

- ▶ on calcule entre 0 Hz et 4700 Hz la différence pour chaque fréquence entre la DSP normalisée obtenue et la vraie enveloppe gaussienne \Rightarrow la somme normalisée donne la distance entre les deux
- ▶ on fait cette mesure un grand nombre de fois : le résultat dépend des phases respectives des partiels
- ▶ on fait cette mesure pour différents ordres p

Algorithme 1 – Levinson-Durbin – Enveloppe

► voir :

scriptenvelopelevinson.m

Algorithme 1 – Levinson-Durbin – Enveloppe – Norme 1

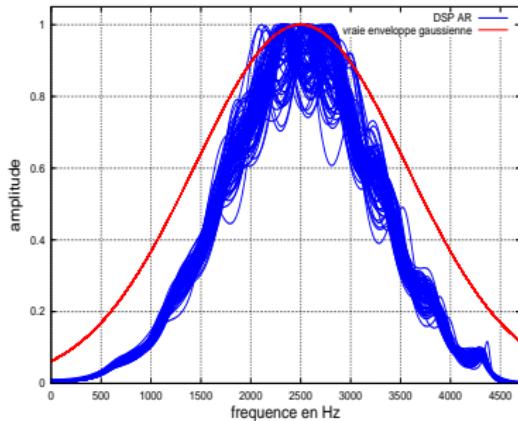
p	N	min	max	mean	σ
4	105	0.304409	0.797658	0.613865	0.124164
8	3155	0.275471	0.628411	0.523677	0.050609
30	615	0.239516	0.433474	0.330744	0.036386
40	600	0.258293	0.428611	0.342025	0.030023
50	635	0.231126	0.425893	0.318209	0.028729
60	600	0.239586	0.419592	0.327151	0.030209
100	605	0.244789	0.441213	0.337577	0.033792
120	3000	0.279712	0.659216	0.390358	0.034345

Algorithme 1 – Levinson-Durbin – Enveloppe

- ▶ il y a un souci avec la normalisation de la DSP AR, l'amplitude des DSP est beaucoup trop grande (voir le script)
- ▶ on sait que les amplitudes des pics ne correspondent pas aux amplitudes réelles des sinus (voir l'exemple de *mylevinsondurbin.m*)
- ▶ la normalisation utilisée pour obtenir les précédents résultats consistait seulement à mettre le maximum de la DSP AR à 1 (**en pratique, par FFT, on pourrait estimer le maximum de l'enveloppe réelle, donc c'est d'accord**) [norme 1]
- ▶ sur la figure suivante, on voit que ça amène à sous-estimer l'amplitude de l'enveloppe
- ▶ on pourrait normaliser la surface de la DSP AR entre 0 Hz et 4700 Hz pour qu'elle corresponde à celle de la vraie DSP (**en pratique, on n'a pas facilement accès à cette information**) [norme 2] 

Algorithme 1 – Levinson-Durbin – Enveloppe – Norme 1

$p = 50$:



Algorithme 1 – Levinson-Durbin – Enveloppe – Norme 2

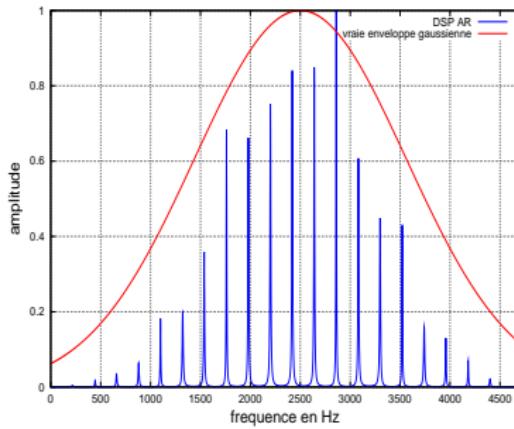
p	N	min	max	mean	σ
4	100	0.367924	0.913593	0.597735	0.125032
8	100	0.367619	0.565783	0.466286	0.039968
30	100	0.284529	0.335381	0.311780	0.009849
40	100	0.275204	0.329546	0.308516	0.009960
50	100	0.276665	0.325316	0.306041	0.008852
60	100	0.273146	0.329655	0.305185	0.011330
100	100	0.282910	0.333140	0.307146	0.010341
120	100	0.279896	0.381355	0.318117	0.013607

Utiliser la deuxième normalisation permet de réduire un peu l'erreur, et d'élargir la plage pour laquelle les enveloppes sont acceptables

Algorithme 1 – Levinson-Durbin – Enveloppe – Norme 1

Est-ce qu'en augmentant suffisamment l'ordre p on arrive à récupérer les 20 partiels ?

Oui ! Ainsi, pour $p = 200$, on obtient par exemple :



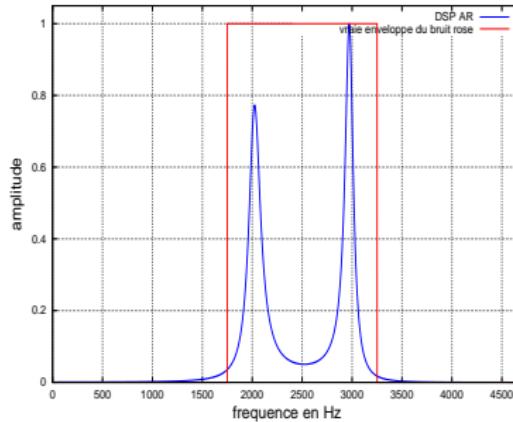
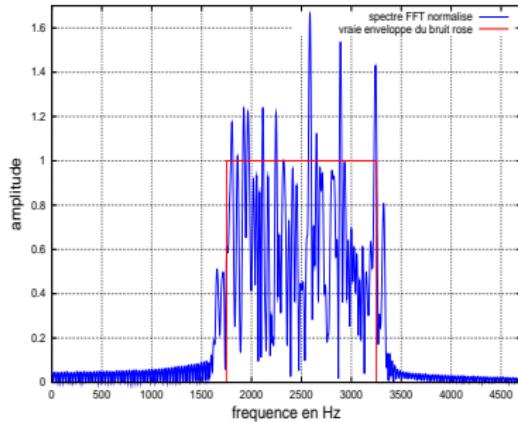
Algorithme 1 – Levinson-Durbin – Bruit rose

Le bruit rose considéré :

- ▶ fréquence centrale du bruit rose : $f_c = 2500 \text{ Hz}$
- ▶ largeur de la bande : 1500 Hz
- ▶ l'amplitude du bruit vaut 1 dans la bande, et 0 donc en-dehors
 - ▶ voir transparent suivant
- ▶ on fait d'abord un bruit blanc, qu'on filtre après coup
- ▶ la longueur du signal est toujours 1280 échantillons, et la fréquence d'échantillonnage 32000 Hz

Algorithme 1 – Levinson-Durbin – Bruit rose

Exemple de spectre FFT et d'enveloppe spectrale AR ($p = 10$, norme 1) obtenus avec le signal précédent :



Algorithme 1 – Levinson-Durbin – Bruit rose

Note : la normalisation du spectre FFT est parfaitement contrôlée (considérations énergétiques concernant les spectres continus)

Mesures :

- ▶ on calcule entre 0 Hz et 4700 Hz la différence pour chaque fréquence entre la DSP normalisée obtenue et la vraie enveloppe du bruit rose \Rightarrow la somme normalisée donne la distance entre les deux
- ▶ on fait cette mesure un grand nombre de fois : le résultat dépend du tirage initial du bruit blanc
- ▶ on fait cette mesure pour différents ordres p

Algorithme 1 – Levinson-Durbin – Bruit rose

► voir :

scriptpinklevinson.m

Algorithme 1 – Levinson-Durbin – Bruit rose – Norme 3

Norme 3 : DSP pas normalisée en amplitude

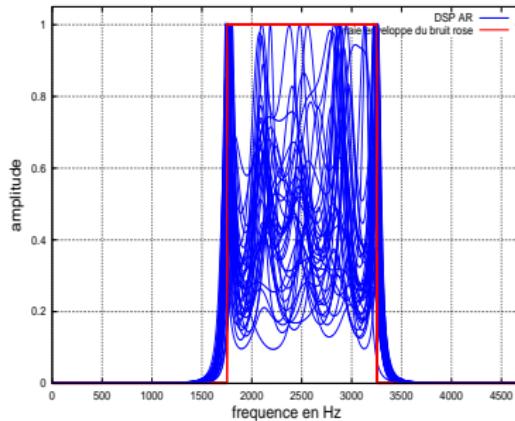
p	N	min	max	mean	σ
4	100	0.819920	1.414590	1.074078	0.093108
8	100	0.379378	1.142463	0.723965	0.155932
20	100	0.287076	0.872924	0.526497	0.119039
30	100	0.229081	0.732262	0.378158	0.089417
40	100	0.225694	0.626082	0.396875	0.081685
50	100	0.251469	0.702591	0.417946	0.090355
60	100	0.260456	0.672508	0.401787	0.081202
100	100	0.286943	0.678895	0.449489	0.086145
120	100	0.283268	0.736525	0.477500	0.090514

Algorithme 1 – Levinson-Durbin – Bruit rose

- ▶ pour obtenir les résultats précédents, aucune normalisation de l'enveloppe spectrale n'a été utilisée [norme 3]
- ▶ la normalisation utilisée pour obtenir les résultats suivants consiste à mettre le maximum de la DSP AR à 1 (**en pratique, par FFT, on a un peu de mal à estimer le maximum de l'enveloppe réelle**)
[norme 1]
 - ▶ sur la figure suivante, on voit que ça amène comme précédemment à sous-estimer l'amplitude de l'enveloppe
- ▶ ou à normaliser la surface de la DSP AR entre 0 Hz et 4700 Hz pour qu'elle corresponde à celle de la vraie DSP (**en pratique, on n'a pas facilement accès à cette information**) [norme 2]

Algorithme 1 – Levinson-Durbin – Bruit rose – Norme 1

$p = 50$:



Algorithme 1 – Levinson-Durbin – Bruit rose

Utiliser la normalisation 2 permet de réduire notablement l'erreur, contrairement à ce qui se passe pour l'enveloppe de raies spectrales :

p	min	max	mean	σ
4	0.7580	0.8837	0.8236	0.0262
8	0.4413	0.8691	0.6985	0.0861
20	0.3922	0.854	0.6403	0.0963
30	0.3000	0.8203	0.5683	0.1151
40	0.2687	0.8085	0.5755	0.1091
50	0.3502	0.8178	0.5802	0.1027
60	0.2805	0.8207	0.5743	0.1064
100	0.3968	0.8855	0.6292	0.0969
120	0.4235	0.8581	0.6492	0.0868

max de l'enveloppe à 1
Norme 1 ($N = 100$)

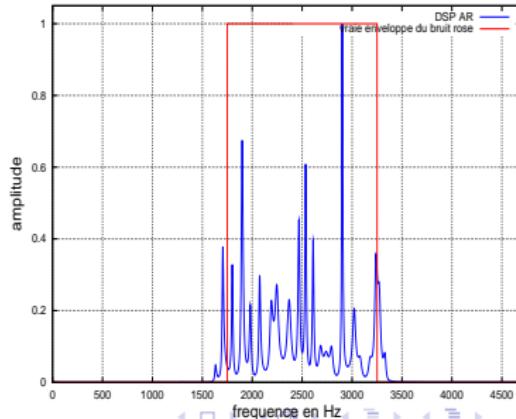
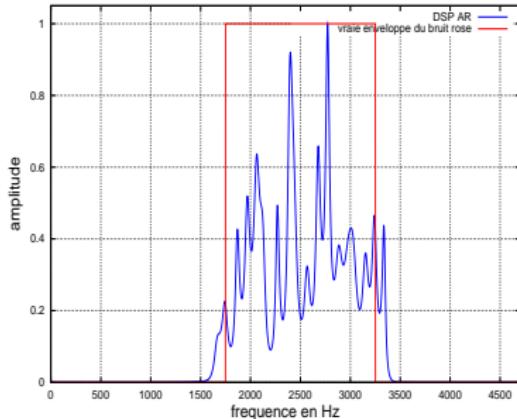
min	max	mean	σ
0.9186	1.1450	1.0332	0.0509
0.3924	0.9246	0.6696	0.1293
0.2556	0.7796	0.5157	0.1164
0.2065	0.5735	0.367	0.0792
0.213	0.5423	0.3745	0.0692
0.2191	0.5502	0.3843	0.0676
0.2663	0.5666	0.3859	0.0682
0.2823	0.5779	0.4382	0.0683
0.2689	0.6794	0.4658	0.076

surface de l'enveloppe normalisée
Norme 2 ($N = 100$)

Algorithme 1 – Levinson-Durbin – Bruit rose – Norme 1

Et si on prend un ordre p extrêmement grand, qu'est-ce qui se passe ?

Pour $p = 300$ et $p = 500$, on obtient par exemple :



Algorithmes – Résumé des résultats

	Levinson-Durbin	FFT
1 sinusoïde	$p \geq 4$	oui
2 sinusoïdes	$p = 250 : 63\%$ $p = 265 : 88\%$ $p = 300 : 100\%$	Blackman 32 % Hanning 40 % (Rectangulaire 64 % méth. autre)
Enveloppe	norme 1 : $e > 0.318$ <i>norme 2</i> : $e > 0.305$	détection des pics à faire
20 partiels	oui ($p = 200$)	oui
bruit rose	<i>norme 1</i> : $e > 0.568$ <i>norme 2</i> : $e > 0.367$ <i>norme 3</i> : $e > 0.378$	$e \simeq 0.5^1$

¹voir scriptpinkfft.m

Algorithme 1 – Levinson-Durbin – Les amplitudes (norme 3)

- ▶ *mylevinsondurbin.m* : l'amplitude du sinus est 1, et on obtient un pic gigantesque (amplitude de l'ordre de 25-100)
- ▶ *scriptdoublonlevinson.m* : l'amplitude de chaque sinus vaut 1, et on obtient des pics gigantesques (amplitudes quasi égales, et de l'ordre de 250)
- ▶ *scriptenveloplevinson.m* : l'amplitude du plus grand partiel est 1, et on obtient un maximum gigantesque (de l'ordre de 40)
- ▶ *scriptpinklevinson.m* : l'amplitude dans la bande du bruit rose est 1, et on obtient une DSP d'amplitude raisonnable (même si chahutée)

Algorithme 1 – Levinson-Durbin – Le code matlab/octave

- ▶ j'ai mis le code de *mylevinsondurbin.m* et de *scriptphaselevinson.m* sur mon site :

[http://www.metz.supelec.fr/metz/personnel/
rossignol/mylevinsondurbin.m](http://www.metz.supelec.fr/metz/personnel/rossignol/mylevinsondurbin.m)

[http://www.metz.supelec.fr/metz/personnel/
rossignol/scriptphaselevinson.m](http://www.metz.supelec.fr/metz/personnel/rossignol/scriptphaselevinson.m)

- ▶ n'oubliez pas qu'à un moment donné je vais vous demander de produire des DSP, et puis que je vais vous lâcher dans la nature en ce qui concerne leur interprétation

Algorithme 1 – Levinson-Durbin – Le code en C

- ▶ dans une perspective d'utilisation de ces méthodes d'analyse spectrale en temps réel par exemple, il faudrait traduire ce code en C (par exemple)
- ▶ si on examine le code matlab/octave, on voit qu'il n'y a rien de compliqué, à part à la rigueur (et encore) l'autocorrélation
 - ▶ sauf si pour des raisons de rapidité il fallait la calculer via la FFT (note : je fournis une bibliothèque minimalistique de calcul de FFT en C ⇒ lien sur mon compte)
- ▶ donc, pas besoin a priori d'utiliser de bibliothèques pré-existantes par exemple pour le calcul matriciel, etc.
- ▶ bien sûr, il faudrait aussi considérer tous les problèmes liés à l'acquisition des signaux

Ce qu'il faudrait aussi caractériser

- ▶ une sinusoïde noyée dans du bruit ; le σ^2 du bruit devenant de plus en plus grand
- ▶ le nombre de points N varie ; N devenant de plus en plus petit
- ▶ une sinusoïde dont la fréquence varie continuement (chirp)
 - ▶ linéairement... ou pas ; pour simuler les variations de pitch d'un signal de parole ou le vibrato d'un signal de musique
- ▶ une sinusoïde dont l'amplitude varie continuement
 - ▶ linéairement... ou pas ; pour simuler les variations d'énergie d'un signal de parole (difficultés pour produire un phonème soutenu) ou le trémolo d'un signal de musique
- ▶ et finalement sur un vrai signal, par exemple de parole (extraction des pics et/ou de l'enveloppe spectrale)

Algorithme 1 – Levinson-Durbin – À vous de travailler

- ▶ récupérez le code sur mon site :

<http://www.metz.supelec.fr/metz/personnel/rossignol/mylevinsondurbin.m>

<http://www.metz.supelec.fr/metz/personnel/rossignol/scriptphaselevinson.m>

- ▶ récupérez le son sur mon site :

<http://www.metz.supelec.fr/metz/personnel/rossignol/fluteircam.wav>

- ▶ traitez-le

- ▶ récupérez des spectres de raies et/ou des enveloppes spectrales
- ▶ comparez par rapport aux méthodes d'analyse spectrale non-paramétrique
- ▶ interprétez vos résultats !
- ▶ faites profiter le rapport que Laurant Duval vous a demandé d'écrire de tout votre travail

Algorithme 2 – Burg

Page intentionnellement laissée blanche

Algorithme 2 – Burg

- ▶ cette fois, on n'utilise plus les coefficients d'autocorrélation estimés, mais le fait que la modélisation AR et la prédiction linéaire sont liées
- ▶ en effet, on peut prouver que $\hat{x}_n = - \sum_{k=1}^p a_k x_{n-k}$ est la meilleure estimée de x_n , dans le cadre de la prédiction linéaire (voir les références)
- ▶ l'erreur de prédiction à l'instant n est alors ($p + 1 \leq n \leq N$) :

$$e_p(n) = x_n - \hat{x}_n = \sum_{k=0}^p a_k x_{n-k} \text{ (avec } a_0 = 1\text{)}$$

Algorithme 2 – Burg

- ▶ $e_p(n)$ est l'erreur de prédiction **avant** : on prédit x_n à partir des échantillons du passé $\{x_{n-p} \dots x_{n-1}\}$
- ▶ on peut définir de la même façon la prédiction **arrière** $r_p(n)$: on prédit (*<< postdit >>*) x_{n-p} à partir des échantillons du futur $\{x_{n-p+1} \dots x_n\}$
- ▶ on a du coup ($p + 1 \leq n \leq N$) :

$$r_p(n) = x_{n-p} - \hat{x}_{n-p} = \sum_{k=0}^p a_k x_{n-p+k} \text{ (avec } a_0 = 1\text{)}$$

Algorithme 2 – Burg

- ▶ toute l'idée de l'algorithme de Burg consiste maintenant à minimiser la somme des deux erreurs de prédiction mises au carré
- ▶ notez qu'elles sont toutes les deux définies entre $n = p + 1$ et $n = N$
- ▶ de plus, l'algorithme de Burg est itératif, comme celui de Levinson-Durbin : on augmente l'ordre progressivement
- ▶ on obtient de nouveau les ensembles de paramètres :
 $\{a_{1,1}, \sigma_1^2\}$, $\{a_{2,1}, a_{2,2}, \sigma_2^2\}$, ..., $\{a_{p,1}, a_{p,2}, \dots, a_{p,p}, \sigma_p^2\}$

Algorithme 2 – Burg

- ▶ donc, on veut donc minimiser cette somme :

$$E_p = \sum_{n=p+1}^N \{ e_p^2(n) + r_p^2(n) \}$$

- ▶ note : si on se contente de minimiser $\sum_{n=p+1}^N e_p^2$, on retombe sur l'algorithme de Levinson ; et d'ailleurs c'est en passant par ça qu'on prouve aussi l'assertion << la meilleure estimée de x_n dans le cadre de la prédition linéaire est $- \sum_{k=1}^p a_k x_{n-k}$ >>

Algorithme 2 – Burg

- ▶ reprenons maintenant la récursivité de Levinson-Durbin, qui dit (k étant l'ordre courant : il augmente récursivement) :

$$a_{k,k} = -\frac{\hat{R}_{xx}(k) + \sum_{l=1}^{k-1} a_{k-1,l} \hat{R}_{xx}(k-l)}{\sigma_{k-1}^2}$$

$$a_{k,i} = a_{k-1,i} + a_{k,k} a_{k-1,k-i}^* \text{ pour } i < k$$

- ▶ en posant $\gamma_{k+1} = \frac{\sum_{l=0}^k a_{k,l} \hat{R}_{xx}(k+1-l)}{\sigma_k^2}$ (γ prend une place importante), on l'écrit plus facilement ainsi :

$$a_{k,k} = -\gamma_k \text{ F1}$$

$$a_{k,i} = a_{k-1,i} - \gamma_k a_{k-1,k-i}^* \text{ pour } i < k \text{ F2}$$

Algorithme 2 – Burg

- en manipulant quelque peu les équations des erreurs de prédition avant et arrière, ainsi que les équations de récursivité de Levinson-Durbin, on parvient à obtenir :

$$e_k(n) = e_{k-1}(n) - \gamma_k r_{k-1}(n-1) \quad F3$$

$$r_k(n) = r_{k-1}(n-1) - \gamma_k e_{k-1}(n) \quad F4$$

- en substituant ces équations dans l'équation que l'on a obtenue pour E_p , on peut écrire :

$$E_k = D_k \gamma_k^2 - 2N_k \gamma_k + D_k$$

que l'on va vouloir minimiser par rapport à γ_k (ces γ correspondent aux inverses des coefficients de réflexion, bien sûr)

Algorithme 2 – Burg

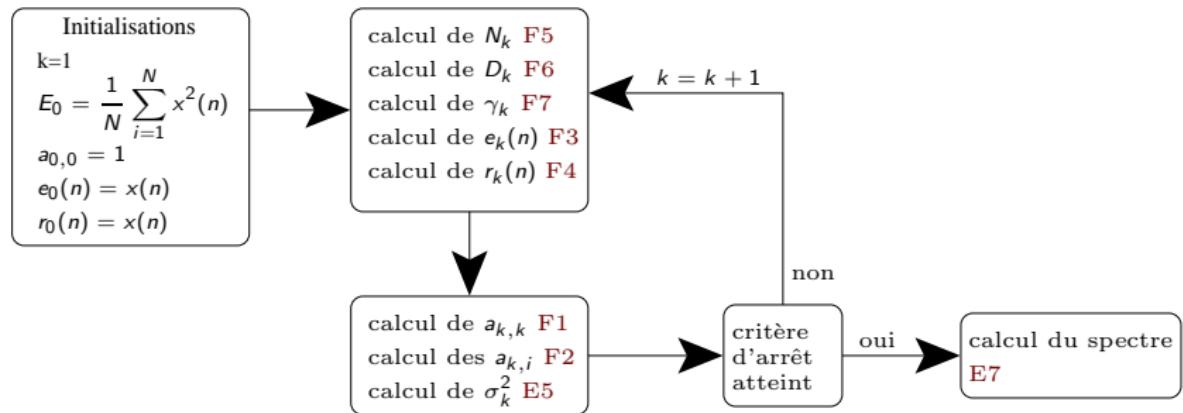
- ▶ et on a pour N_k et D_k :

$$N_k = 2 \sum_{n=k+1}^N e_{k-1}(n) r_{k-1}(n-1) \quad \text{F5}$$

$$\text{et } D_k = \sum_{n=k+1}^N \{e_{k-1}^2(n) + r_{k-1}^2(n-1)\} \quad \text{F6}$$

- ▶ il reste à dériver E_k par rapport à γ_k et à annuler cette dérivée
- ▶ on obtient aisément : $\gamma_k = \frac{N_k}{D_k} \quad \text{F7}$
- ▶ et on a tout ce qu'il faut pour trouver les paramètres AR

Algorithme 2 – Burg



Algorithme 2 – Burg – Code matlab

```
function [aa, sigma2, kk, ff, mydsp] = myburg(xx, pp, fe)

% Initialisations
N = length(xx);
xx = xx( :);
ef = xx;
eb = xx;
aa = 1;
EE = xx'*xx./N ;
kk = zeros(1, pp);% Pré-allocation de 'kk' (pour améliorer la vitesse d'exécution)

for tt=1 :pp
    % Calcule le coefficient de réflexion suivant
    efp = ef(2 :end);
    ebp = eb(1 :end-1);
    num = -2.*ebp'*efp; %%% c'est -Nk (F5)
    den = efp'*efp+ebp'*ebp; %%% c'est Dk (F6)
    kk(1,tt) = num./den; %%% c'est  $-\gamma$  dans le cours (F7)
    % Mise à jour des erreurs de prédition 'avant' et 'arrière'
    ef = efp + kk(tt)*ebp; %%% (F3)
    eb = ebp + kk(tt)'*efp; %%% (F4)
    aa=[aa ;0] + kk(1,tt)*[0;conj(flipud(aa))];% Mise à jour des coefficients AR (F1 et F2)
    EE(tt+1) = (1 - kk(1,tt)'*kk(1,tt))*EE(tt);% Mise à jour de l'erreur de prédition globale (E5)
end
aa = aa( :);
```



Algorithme 2 – Burg – Premiers tests

On va faire quelques tests, avec des signaux (sons) simples pour vérifier qu'on obtient bien des densités spectrales de puissance intéressantes. Les 4 tests qu'on considère ici sont :

- ▶ on a une sinusoïde pure \Rightarrow est-ce qu'on arrive à la mettre en évidence ?
- ▶ on a deux sinusoïdes proches en fréquence \Rightarrow est-ce qu'on arrive à les discriminer ?
- ▶ on a un signal compliqué : une somme de sinusoïdes harmoniques \Rightarrow est-ce qu'on arrive à obtenir une enveloppe spectrale, modélisant ce signal, et/ou les n partiels ?
- ▶ on a un signal compliqué : un bruit rose \Rightarrow est-ce qu'on arrive à obtenir une enveloppe spectrale, modélisant ce signal ?

Algorithme 2 – Burg – Une sinusoïde

myburg.m

Paramètres de l'exemple :

- ▶ $f_e = 32000$ Hz
- ▶ signal : une sinusoïde présente
 - ▶ de fréquence 440 Hz
 - ▶ et de phase aléatoire, tirée entre 0 et 2π
 - ▶ de taille 1280 échantillons (40 ms)
- ▶ $p = 4$

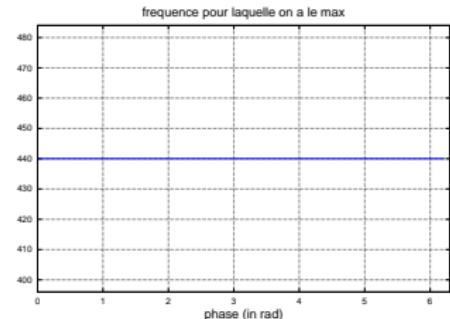
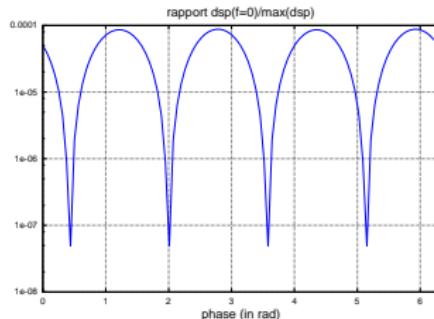
Algorithme 2 – Burg – Une sinusoïde

scriptphaseburg.m

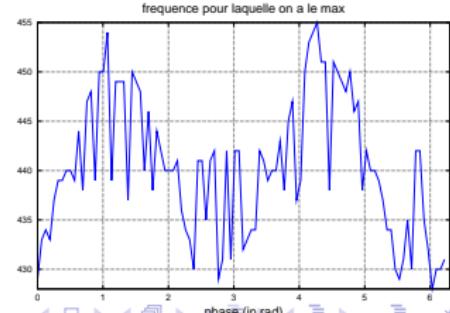
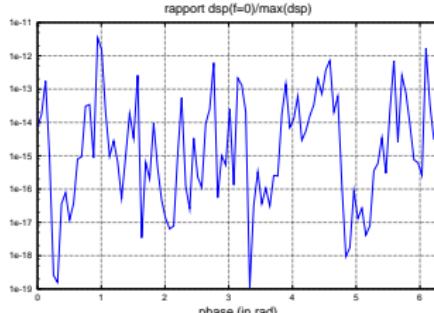
- ▶ on regarde ce qui se passe en fonction de la phase de la sinusoïde
- ▶ on mesure le rapport entre l'amplitude de la DSP en $f = 0$ et l'amplitude maximum de la DSP (sensée se trouver à $\pm 440 \text{ Hz}$)
- ▶ on regarde où se trouve en réalité l'amplitude maximum de la DSP
- ▶ ce pour différents ordres : $p = 1, p = 2, p = 3, p = 4, \dots$

Algorithme 2 – Burg – Une sinusoïde

$p = 2 :$



$p = 20 :$



Algorithme 2 – Burg – Une sinusoïde

Dans le tableau suivant, on donne le nombre de fois (sur 100) où la fréquence du maximum a été trouvée entre 400 Hz et 480 Hz.

p	1	2	3	4	20	30	50	100	150
%	0	100	100	100	100	100	100	100	100

Un ordre trop élevé ne semble jamais devenir problématique.

Algorithme 2 – Burg – Intérêt

- ▶ est-ce que l'algorithme de Burg présente le même intérêt que celui de Levinson-Durbin en termes de haute-résolution ?
- ▶ on refait les mêmes tests qu'avec Levinson-Durbin
 - ▶ 2 sinusoïdes proches, séparées de 26 Hz (440 Hz et 466 Hz) et de même amplitude, etc. : voir transparent suivant
- ▶ et on fait varier l'ordre p
 - ▶ cette fois, en partant de petits ordres

Algorithme 2 – Burg – Intérêt

- ▶ la longueur du signal est 1280 échantillons (une seule trame de 40 ms)
- ▶ sinusoïdes de fréquences 440 Hz et 466 Hz
- ▶ la DSP est calculée sur 32768 points
- ▶ méthode de détection simple : on détecte les maximums et minimums locaux entre $f_{min} - 40 \text{ Hz}$ et $f_{max} + 40 \text{ Hz}$
- ▶ puis on vérifie qu'il y a bien 2 maximums locaux, 1 minimum local, et que celui-ci se situe entre les 2 maximums locaux
- ▶ le résultat obtenu dépend de la phase respective des 2 sinus
- ▶ il dépend aussi de l'ordre du modèle AR

Algorithme 2 – Burg – Intérêt

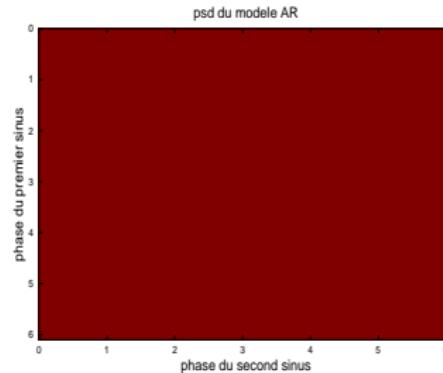
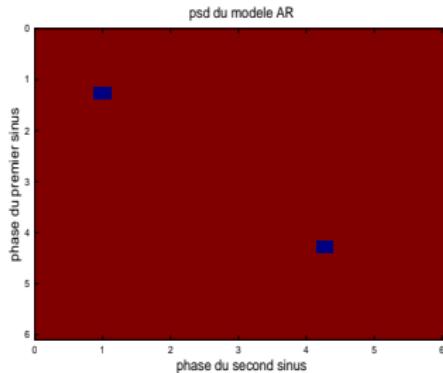
► voir :

scriptdoublonburg.m

Algorithme 2 – Burg – Intérêt

Burg (ordre : 4) : 99.68 % de succès

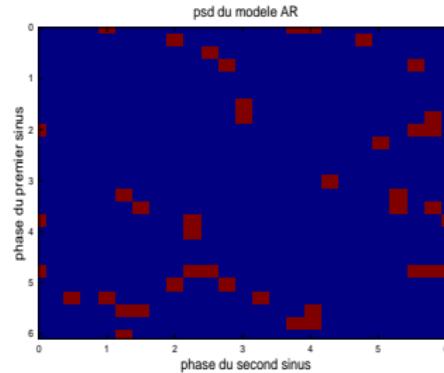
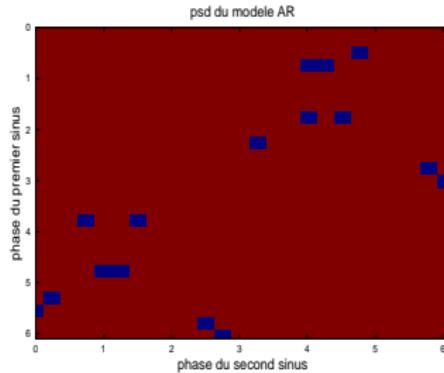
Burg (ordre : 5) : 100 % de succès (apparition parfois d'un pic parasite autour de $f = 0$)



Algorithme 2 – Burg – Intérêt

Burg (ordre : 6) : 97.44 % de succès (apparition fréquente d'un pic parasite autour de $f = 0$)

Burg (ordre : 7) : 6.72 % de succès (début d'instabilité)



Algorithme 2 – Burg – Intérêt

- ▶ ça marche même pour de petits ordres p
- ▶ notez que $p = 4$ est théoriquement l'ordre minimum qui permet de détecter 2 sinusoïdes (notre cas ici)
- ▶ de ce fait, le calcul du modèle AR de Burg est beaucoup plus rapide que celui du calcul du modèle AR de Levinson-Durbin
- ▶ on atteint comme pour l'algorithme de Levinson-Durbin 100 % de bonnes détections
- ▶ on fait encore une fois mieux qu'avec la FFT, pour laquelle, en tirant sur la corde, on obtient au mieux 64 % de bonnes détections

Algorithme 2 – Burg – Intérêt

- ▶ de plus, dès que l'ordre devient trop grand, il y a instabilité
 - ▶ apparitions de pics parasites
 - ▶ les vrais pics tendent à se dédoubler (c'est un problème de la méthode bien connu)
- ▶ ça rendra la mise en place de méthodes d'estimation de l'ordre p à utiliser nécessaire
- ▶ la plage d'ordres p pour lesquels la méthode marche bien n'est quand même pas très large

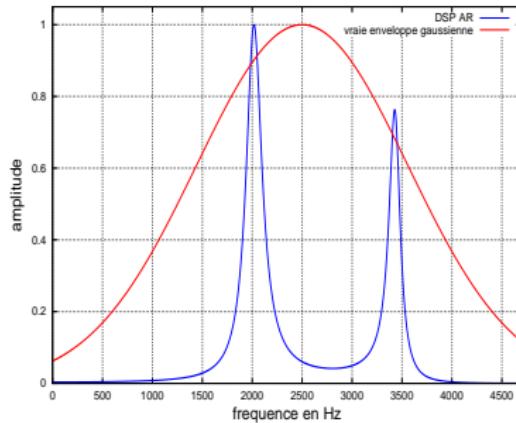
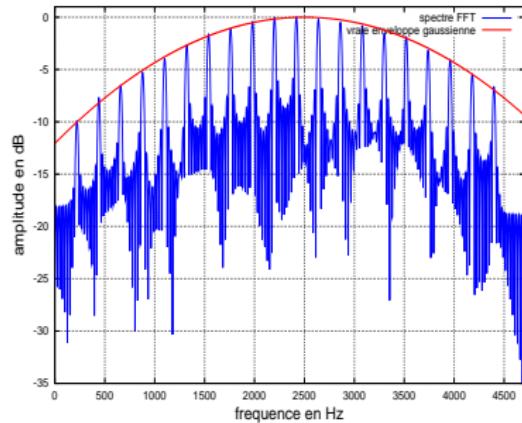
Algorithme 2 – Burg – Enveloppe

On a un signal harmonique (composé d'une somme de partiels harmoniques). Le signal considéré :

- ▶ fréquence fondamentale : $f_0 = 220 \text{ Hz}$
- ▶ nombre de partiels : 20
- ▶ amplitudes des partiels : elles suivent une gaussienne centrée en $f_{max} = 2500 \text{ Hz}$ et d'écart-type 1500
 - ▶ voir transparent suivant
- ▶ phases des partiels : tirées aléatoirement entre 0 et 2π
- ▶ la longueur du signal est toujours 1280 échantillons, et la fréquence d'échantillonnage 32000 Hz

Algorithme 2 – Burg – Enveloppe

Exemple de spectre FFT et d'enveloppe spectrale AR ($p = 4$, norme 1) obtenus avec le signal précédent :



Algorithme 2 – Burg – Enveloppe – Norme 1

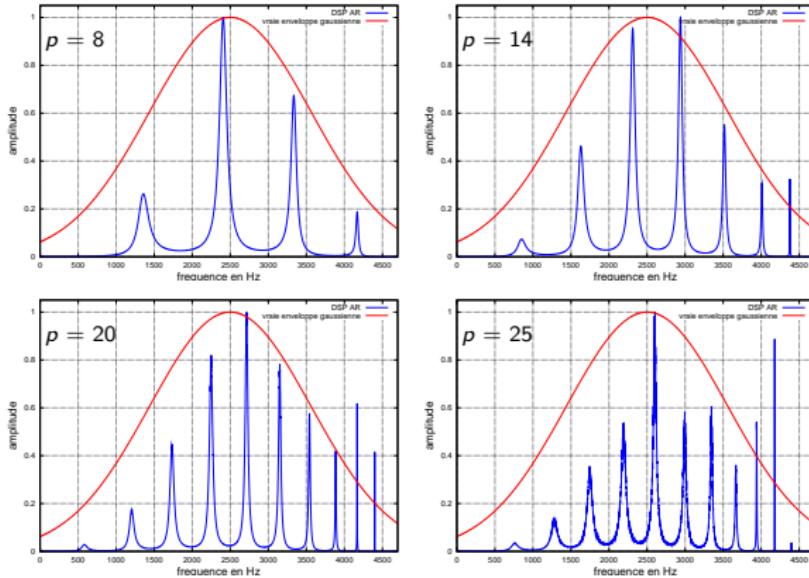
► voir :

scriptenvelopburg.m

Algorithme 2 – Burg – Enveloppe – Norme 1

- ▶ pour $p = 4$, on récupère déjà des résonances fines
- ▶ plus l'ordre p va augmenter, bien sûr plus on aura de fines résonances (voir le transparent suivant)
- ▶ on ne mesure même pas la distance entre l'enveloppe AR et l'enveloppe vraie, comme on avait fait pour la méthode de Levinson-Durbin
- ▶ la méthode de Burg ne semble pas trop adaptée pour l'estimation des enveloppes spectrales de signaux harmoniques
- ▶ ou alors il faudrait comme pour le spectre FFT mettre en place une étape supplémentaire de détection des pics

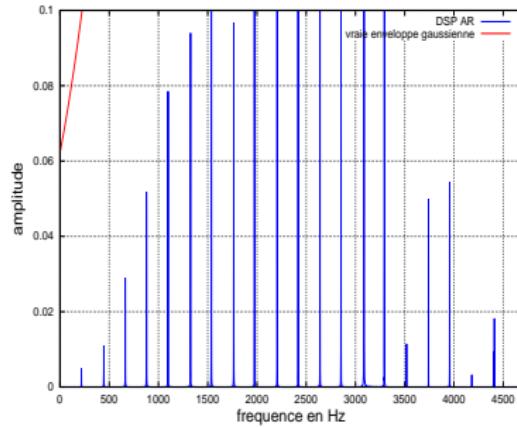
Algorithme 2 – Burg – Enveloppe



Algorithme 2 – Burg – Enveloppe

Est-ce qu'en augmentant suffisamment l'ordre p on arrive à récupérer les 20 partiels ?

Oui ! Pour $p = 100$, on obtient par exemple (en zoomant en y) :



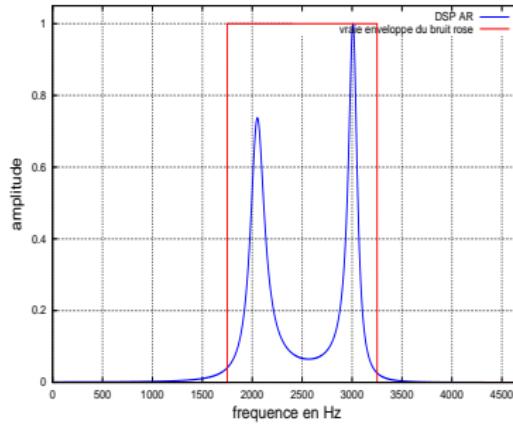
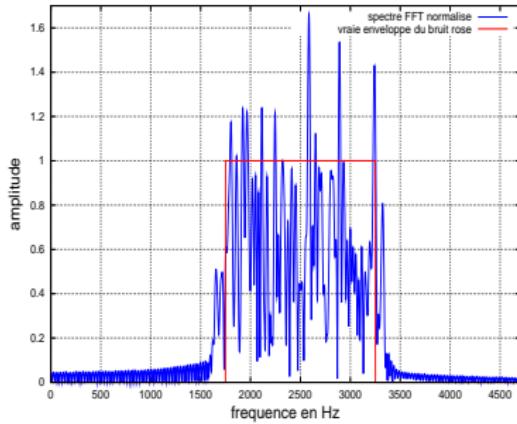
Algorithme 2 – Burg – Bruit rose

Le bruit rose considéré :

- ▶ fréquence centrale du bruit rose : $f_c = 2500 \text{ Hz}$
- ▶ largeur de la bande : 1500 Hz
- ▶ l'amplitude du bruit vaut 1 dans la bande, et 0 donc en-dehors
 - ▶ voir transparent suivant
- ▶ on fait d'abord un bruit blanc, qu'on filtre après coup
- ▶ la longueur du signal est toujours 1280 échantillons, et la fréquence d'échantillonnage 32000 Hz

Algorithme 2 – Burg – Bruit rose

Exemple de spectre FFT et d'enveloppe spectrale AR ($p = 10$, norme 1) obtenus avec le signal précédent :



Algorithme 2 – Burg – Bruit rose

Note : la normalisation du spectre FFT est parfaitement contrôlée
(considérations énergétiques concernant les spectres continus)

Mesures :

- ▶ on calcule entre 0 Hz et 4700 Hz la différence pour chaque fréquence entre la DSP normalisée obtenue et la vraie enveloppe du bruit rose \Rightarrow la somme normalisée donne la distance entre les deux
- ▶ on fait cette mesure un grand nombre de fois : le résultat dépend du tirage initial du bruit blanc
- ▶ on fait cette mesure pour différents ordres p

Algorithme 2 – Burg – Bruit rose

► voir :

scriptpinkburg.m

Algorithme 2 – Burg – Bruit rose – Norme 3

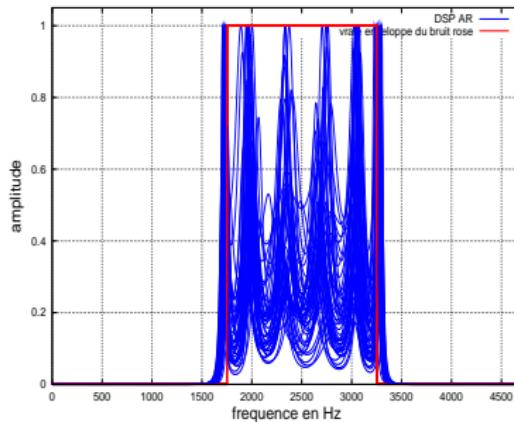
p	N	min	max	mean	σ
4	100	0.816018	1.247149	0.987615	0.083261
8	100	0.759103	1.260950	0.961112	0.091501
12	100	0.325692	0.750349	0.498926	0.094605
20	100	0.302352	0.703861	0.492532	0.091068
40	100	0.289402	0.677849	0.520154	0.086354
50	100	0.361261	0.757746	0.573926	0.087981
60	100	0.401146	0.794777	0.583306	0.078862
100	100	0.340993	0.703464	0.505795	0.082516
120	100	0.350585	0.765132	0.547785	0.086922

Algorithme 2 – Burg – Bruit rose

- ▶ pour obtenir les résultats précédents, aucune normalisation de l'enveloppe spectrale n'a été utilisée
- ▶ la normalisation utilisée pour obtenir les résultats suivants consiste à mettre le maximum de la DSP AR à 1 (**en pratique, par FFT, on a un peu de mal à estimer le maximum de l'enveloppe réelle**)
[Norme 1]
 - ▶ sur la figure suivante, on voit que ça amène à sous-estimer l'amplitude de l'enveloppe
- ▶ ou à normaliser la surface de la DSP AR entre 0 *Hz* et 4700 *Hz* pour qu'elle corresponde à celle de la vraie DSP (**en pratique, on n'a pas facilement accès à cette information**) [Norme 2]

Algorithme 2 – Burg – Bruit rose – Norme 1

$p = 50$:



Algorithme 2 – Burg – Bruit rose

Utiliser la deuxième normalisation permet de réduire notablement l'erreur

p	min	max	mean	σ
4	0.685	0.8554	0.7829	0.034
8	0.7486	0.8763	0.8126	0.0257
20	0.5370	0.9301	0.7104	0.0809
30	0.5201	0.9186	0.7344	0.0925
40	0.4629	0.9042	0.7222	0.1012
50	0.4952	0.9319	0.7521	0.1037
60	0.4864	0.9178	0.7566	0.0801
100	0.506	0.9177	0.7077	0.0824
150	0.5052	0.9338	0.7684	0.0782

max de l'enveloppe à 1

Norme 1 ($N = 100$)

	min	max	mean	σ
0.7834	1.1070	0.9265	0.0704	
0.7826	1.0302	0.9023	0.0513	
0.2758	0.7060	0.4561	0.0844	
0.3563	0.6946	0.501	0.0771	
0.2597	0.7432	0.4809	0.0972	
0.353	0.7203	0.5287	0.0805	
0.3451	0.6736	0.5309	0.0668	
0.3472	0.6999	0.4865	0.0687	
0.3414	0.747	0.5523	0.0765	

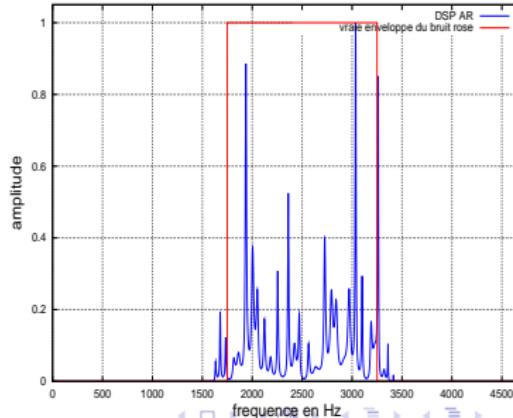
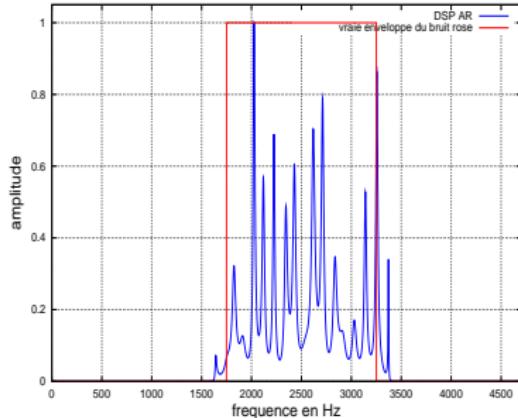
surface de l'enveloppe normalisée

Norme 2 ($N = 100$)

Algorithme 2 – Burg – Bruit rose – Norme 1

Et si on prend un ordre p extrêmement grand, qu'est-ce qui se passe ?

Pour $p = 300$ et $p = 500$, on obtient par exemple :



Algorithmes – Résumé des résultats

	Levinson-Durbin	Burg	FFT
1 sinusoïde	$p \geq 4$	$p \geq 2$	oui
2 sinusoïdes	$p = 250 : 63\%$ $p = 265 : 88\%$ $p = 300 : 100\%$	$p = 4 : 99.68\%$ $p = 5 : 100\%$ $p = 6 : 97\%$ $p = 7 : 7\%$	Blackman 32 % Hanning 40 % (Rectangulaire 64 % méth. autre)
Enveloppe	norm1 : $e > 0.318$ norm2 : $e > 0.305$	pas très efficace	détection des pics à faire
20 partiels	oui ($p = 200$)	oui ($p = 100$)	oui
bruit rose	norm1 : $e > 0.568$ norm2 : $e > 0.367$ norm3 : $e > 0.378$	norm1 : $e > 0.708$ norm2 : $e > 0.456$ norm3 : $e > 0.493$	$e \simeq 0.5$

Algorithme 2 – Burg – Les amplitudes (norme 3)

- ▶ *myburg.m* : l'amplitude du sinus est 1, et on obtient un pic gigantesque (plusieurs ordres de grandeur couverts)
- ▶ *scriptdoublonburg.m* : l'amplitude de chaque sinus vaut 1, et on obtient des pics gigantesques (amplitudes quasi égales, et de l'ordre de 3000)
- ▶ *scriptenvelopburg.m* : l'amplitude du plus grand partiel est 1, et on obtient un maximum gigantesque (de l'ordre de 15)
- ▶ *scriptpinkburg.m* : l'amplitude dans la bande du bruit rose est 1, et on obtient une DSP d'amplitude raisonnable (même si chahutée)

Algorithme 2 – Burg – Le code matlab/octave

- ▶ j'ai mis le code de *myburg.m* sur mon site :

[http://www.metz.supelec.fr/metz/personnel/
rossignol/myburg.m](http://www.metz.supelec.fr/metz/personnel/rossignol/myburg.m)

- ▶ bien sûr vous pouvez commencer à jouer avec, et à interpréter les résultats obtenus pour les autres tests proposés ; ou avec d'autres signaux (comme ceux venant de Laurent Duval)

Algorithme 2 – Burg – Le code en C

- ▶ dans une perspective d'utilisation de ces méthodes d'analyse spectrale en temps réel par exemple, il faudrait traduire ce code en C (par exemple)
- ▶ si on examine le code matlab/octave, on voit qu'il n'y a rien de compliqué
- ▶ donc, pas forcément besoin d'utiliser de bibliothèques pré-existantes par exemple pour le calcul matriciel, etc.
- ▶ bien sûr, il faudrait aussi considérer tous les problèmes liés à l'acquisition des signaux

Algorithme 3 – Marple

Page intentionnellement laissée blanche

Algorithme 3 – Marple

- ▶ cette fois, on ne va pas regarder l'algorithme en détails : on pourrait y passer au moins une séance
- ▶ donc, pour les personnes intéressées, voir la 3ème référence
- ▶ on ne va examiner que les points importants, c'est-à-dire ce qui fait la différence par rapport aux deux précédents algorithmes
- ▶ c'est encore un algorithme itératif : on part de l'ordre 1, et l'on va jusqu'à l'ordre désiré p , en obtenant tous les paramètres AR aux ordres intermédiaires k

Algorithme 3 – Marple

- ▶ **point 1.** : déjà, pour Burg, on a vu qu'on annulait la dérivée de E_k par rapport à γ_k
- ▶ γ_k est le coefficient de réflexion obtenu à l'ordre $k \Rightarrow$ il est égal à $-a_{k,k}$
- ▶ ici, on annule la dérivée de E_k par rapport à tous les paramètres $a_{k,i}$ obtenus à l'ordre k
- ▶ on obtient un système du type : $T_k \mathbf{a}_k = \mathbf{e}_k$, qu'il faut résoudre
- ▶ note 1 : bien sûr, les vecteurs \mathbf{a}_k et \mathbf{e}_k suivent toujours les mêmes notations (slide 71 pour \mathbf{e}_k)
- ▶ note 2 : la matrice T_k s'obtient à partir des échantillons du signal x et seulement d'eux

Algorithme 3 – Marple

- ▶ **point 2.** : deux erreurs de prédiction doivent être ajoutées et elles aussi minimisées (sinon, on retombe sur une méthode existante) :

$$E'_k = \sum_{n=k+1}^{N-1} \{ e_k^2(n+1) + r_k^2(n) \}$$

$$E''_k = \sum_{n=k+1}^{N-1} \{ e_k^2(n) + r_k^2(n+1) \}$$

- ▶ on obtient des systèmes du type : $T'_k \mathbf{a}'_k = \mathbf{e}'_k$ et $T''_k \mathbf{a}''_k = \mathbf{e}''_k$, qu'il faut résoudre conjointement au système précédent

Algorithme 3 – Marple

- ▶ toute la suite consiste à développer et manipuler ces équations pour obtenir une récursion du type de celle de Levinson-Durbin, qui soit exploitable (c'est très tordu et très long à faire)
- ▶ **point 3.** : la récursion obtenue est :

$$a_{k,k} = -\gamma_k \text{ G1}$$

$$a_{k,i} = a'_{k-1,i} - \gamma_k a'^{*}_{k-1,k-i} \text{ pour } i < k \text{ G2}$$

- ▶ ce qu'il faut noter, c'est le remplacement des termes $a_{k-1,i}$ par les termes << décalés temporellement >> $a'_{k-1,i}$

Algorithme 3 – Marple – Code matlab (1/9)

```
function [aa, sigma2, kk, ff, mydsp] = mymarple(xx, pp, fe)

%%%% Initialisations
NN = length(xx);
xx = xx(:);
tol1=0.0; %%% tolérance 1 (0.0 : pas de tolérance)
tol2=0.0; %%% tolérance 2 (0.0 : pas de tolérance)
etat = 1; %%% l'algorithme ne converge pas toujours
OrdreMax = pp;
aa(1) = 1.0;
f1=0.0;
for kk=1 :NN
    f1 = f1 + real(xx(kk)*conj(xx(kk)));
end;
e0 = 2.0*f1;
Q1 = 1.0/e0;
Q2 = conj(Q1*xx(1));
ga = Q1*real(xx(1)*conj(xx(1)));
wa = Q1*real(xx(NN)*conj(xx(NN)));
den = 1.0 - ga - wa;
fa = xx(1);
ba = xx(NN);
ha = Q2*conj(xx(NN));
sa = Q2*xx(NN);
va = Q2*conj(xx(1));
```

Algorithme 3 – Marple – Code matlab (2/9)

```
ua = xx(NN)*xx(NN) ;
ua = Q1*ua ;
Q4 = 1.0/den ;
Q5 = 1.0 - ga ;
Q6 = 1.0 - wa ;
ea = e0 * den ;
Q1 = 1.0/ea ;
ca(1) = conj(Q1*xx(1)) ;
da(1) = Q1*xx(NN) ;
MM=1;
c1 = 0.0 ;
for kk=2 :NN
    c1 = c1 + xx(kk)*conj(xx(kk-1)) ;
end ;
c1 = 2.0*c1 ;
ra(1) = c1 ;
aa(2) = -Q1*ra(1) ;
f1 = real(aa(2)*conj(aa(2))) ;
ea = ea*(1.0-f1) ;

if ( f1 > 1.0 )
    etat=3;
end ;
```

Algorithme 3 – Marple – Code matlab (3/9)

```
%%%% début de la boucle principale
while ( (MM < OrdreMax) & (etat==1) )
    %%% mise à jour des erreurs de prédition
    evieux = ea;
    c1 = xx(MM+1);
    for kk=2 :MM+1
        c1 = c1 + xx(MM-kk+2)*aa(kk);
    end;
    fa = c1;
    c1 = xx(NN-MM);
    for kk=1 :MM
        c1 = c1 + xx(NN-MM+kk)*conj(aa(kk+1));
    end;
    ba = c1;
    %%% mise à jour des vecteurs auxiliaires
    Q1 = 1.0/ea;
    Q2 = conj(Q1*fa);
    Q3 = Q1*ba;
```

Algorithme 3 – Marple – Code matlab (4/9)

```
for kk=MM+1 :-1 :2
    ca(kk) = ca(kk-1) + Q2*aa(kk);
    da(kk) = da(kk-1) + Q3*aa(kk);
end;

ca(1) = Q2;
da(1) = Q3;

%%%% mise à jour des scalaires auxiliaires
Q7 = real(sa*conj(sa));

f1 = real(fa*conj(fa));
f2 = real(va*conj(va));
f3 = real(ba*conj(ba));
f4 = real(ua*conj(ua));

c1 = conj(va)*ha*sa;
ga = ga + f1*Q1 + Q4*( f2*Q6 + Q7*Q5 + 2.0*real(c1) );

c1 = conj(sa)*ha*ua;
wa = wa + f3*Q1 + Q4*( f4*Q5 + Q7*Q6 + 2.0*real(c1) );
```

Algorithme 3 – Marple – Code matlab (5/9)

```
ha = 0.0;  
for kk=1 :MM+1  
    ha = ha + conj(xx(NN-MM+kk-1))*ca(kk);  
end;  
  
sa = 0.0;  
for kk=1 :MM+1  
    sa = sa + xx(NN-kk-1)*ca(kk);  
end;  
  
ua = 0.0;  
for kk=1 :MM+1  
    ua = ua + xx(NN-kk-1)*da(kk);  
end;  
  
va = 0.0;  
for kk=1 :MM+1  
    va = va + conj(xx(kk))*ca(kk);  
end;  
  
%%%% mise à jour du dénominateur  
Q5 = 1.0 - ga;  
Q6 = 1.0 - wa;  
den = Q5*Q6 - real(ha*conj(ha));
```

Algorithme 3 – Marple – Code matlab (6/9)

```
if ( den <= 0.0 )
    etat=2;
    MM=MM+1;
    fprintf(1,'\\netat=2 : donnees numeriques mal conditionnees\\n\\n');
    break;
end;

%%%% mise à jour des paramètres de décalage temporel
Q4 = 1.0/den;
Q1 = Q1*Q4;
c1 = ha*fa*ba ;
alpha = 1.0 / (1.0 + Q1*( f1*Q6 + f3*Q5 + 2.0*real(c1)));
ea = alpha * ea;

c1 = Q6*fa ;c1 = c1 + conj(ba*ha) ;c1 = Q4*c1 ;
c2 = conj(Q5*ba) ;c2 = c2 + ha*fa ;c2 = Q4*c2 ;
c3 = Q6*va ;c3 = c3 + ha*sa ;c3 = Q4*c3 ;
c4 = Q5*sa ;c4 = c4 + va*conj(ha) ;c4 = Q4*c4 ;
c5 = Q6*sa ;c5 = c5 + ha*ua ;c5 = Q4*c5 ;
c6 = Q5*ua ;c6 = c6 + sa*conj(ha) ;c6 = Q4*c6 ;

for kk=2 :MM+1
    aa(kk) = aa(kk) + c1*ca(kk) + c2*da(kk) ;
    aa(kk) = alpha*aa(kk) ;
end;
```

Algorithme 3 – Marple – Code matlab (7/9)

```
M2 = MM/2 + 1;  
for kk=1 :M2  
    MK = MM + 2 - kk;  
  
    save1 = conj(ca(kk));  
    save2 = conj(da(kk));  
    save3 = conj(ca(MK));  
    save4 = conj(da(MK));  
  
    ca(kk) = ca(kk) + c3*save3 + c4*save4 ;  
    da(kk) = da(kk) + c5*save3 + c6*save4 ;  
  
    if ( MK !=kk )  
        ca(MK) = ca(MK) + c3*save1 + c4*save2 ;  
        da(MK) = da(MK) + c5*save1 + c6*save2 ;  
    end;  
end;  
  
%%%% mise à jour de l'ordre  
MM=MM+1;
```

Algorithme 3 – Marple – Code matlab (8/9)

```
delta = 0.0;  
for kk=MM :-1 :2  
    c1 = ra(kk-1) - xx(NN-kk+2)*conj(xx(NN-MM+1));  
    ra(kk) = c1 - xx(MM-1)*conj(xx(kk-1));  
  
    delta = delta + ra(kk)*aa(kk);  
end;  
  
c1 = 0.0;  
for kk=2 :(NN-MM+1)  
    c1 = c1 + xx(kk+MM-1)*conj(xx(kk-1));  
end;  
  
ra(1) = 2.0*c1;  
delta = delta + ra(1);  
Q2 = -delta/ea;  
aa(MM+1) = Q2;  
M2 = floor(MM/2);  
for kk=1 :M2  
    MK = MM - kk;  
    save1 = conj(aa(kk+1));  
    aa(kk+1) = aa(kk+1) + Q2*conj(aa(MK+1));
```

Algorithme 3 – Marple – Code matlab (9/9)

```
if ( MK !=kk )
    aa(MK+1) = aa(MK+1) + Q2*save1 ;
end ;
end ;

f1 = real(Q2*conj(Q2)) ;
ea = ea*( 1.0 - f1);

if ( f1 > 1.0 )
    etat=3;
elseif ( (ea/e0) < tol1 )
    etat=4;
elseif ( ((evieux-ea)/evieux) < tol2 )
    etat=5;
end ;
end ; %% fin de la boucle principale

npp=MM ;
if ( etat !=1 )
    npp=MM-1;
end ;
fprintf(1, "%d ",npp);
aa = aa( :).' ;

sigma2 = ea ;
```

Algorithme 3 – Marple – Premiers tests

On va faire quelques tests, avec des signaux (sons) simples pour vérifier qu'on obtient bien des densités spectrales de puissance intéressantes. Les 4 tests qu'on considère ici sont :

- ▶ on a une sinusoïde pure \Rightarrow est-ce qu'on arrive à la mettre en évidence ?
- ▶ on a deux sinusoïdes proches en fréquence \Rightarrow est-ce qu'on arrive à les discriminer ?
- ▶ on a un signal compliqué : une somme de sinusoïdes harmoniques \Rightarrow est-ce qu'on arrive à obtenir une enveloppe spectrale, modélisant ce signal, et/ou les n partiels ?
- ▶ on a un signal compliqué : un bruit rose \Rightarrow est-ce qu'on arrive à obtenir une enveloppe spectrale, modélisant ce signal ?

Algorithme 3 – Marple – Une sinusoïde

mymarple_matlab.m

Paramètres de l'exemple :

- ▶ $f_e = 32000$ Hz
- ▶ signal : une sinusoïde présente
 - ▶ de fréquence 440 Hz
 - ▶ et de phase aléatoire, tirée entre 0 et 2π
 - ▶ de taille 1280 échantillons (40 ms)
- ▶ $p = 4$
- ▶ **attention** : on est obligé d'ajouter un peu de bruit blanc, pour assurer mieux la stabilité de la convergence ; la variance de ce bruit est très petite (le bruit n'est pas visible sur la forme d'onde) : $\sigma^2 = 1e^{-4}$

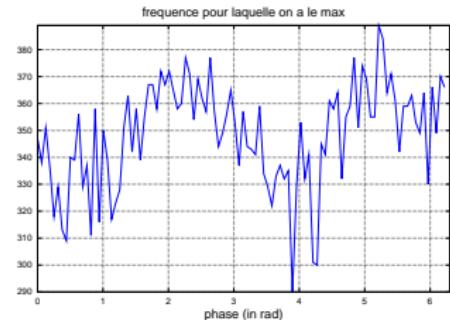
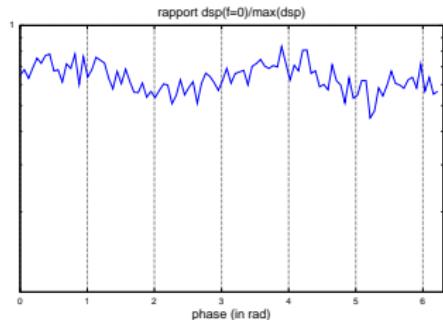
Algorithme 3 – Marple – Une sinusoïde

scriptphasemarple.m

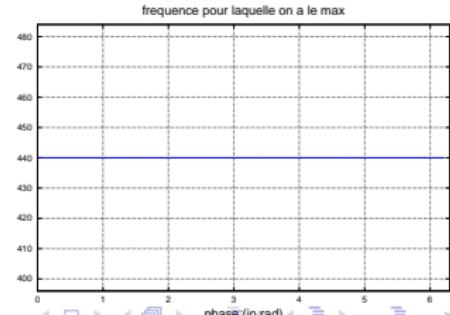
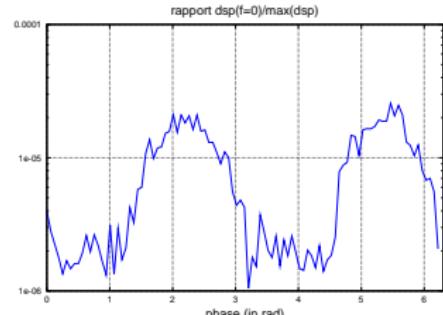
- ▶ on regarde ce qui se passe en fonction de la phase de la sinusoïde
- ▶ on mesure le rapport entre l'amplitude de la DSP en $f = 0$ et l'amplitude maximum de la DSP (sensée se trouver à $\pm 440 \text{ Hz}$)
- ▶ on regarde où se trouve en réalité l'amplitude maximum de la DSP
- ▶ ce pour différents ordres : $p = 2, p = 3, p = 4, \dots$

Algorithme 3 – Marple – Une sinusoïde

$p = 2 :$



$p = 10 :$



Algorithme 3 – Marple – Une sinusoïde

Dans le tableau suivant, on donne le nombre de fois (sur 100) où la fréquence du maximum a été trouvée entre 400 Hz et 480 Hz.

p	2	3	4	10	11	12	15	20	30	50
%	0	100	100	100	98	97	76	69	67	12

Un ordre trop élevé devient problématique aux alentours de 12
En ajoutant un bruit blanc de variance $\sigma^2 = 1e^{-3}$, on obtient :

p	2	3	4	10	11	12	15	20	30	40	50
%	0	0	99	100	100	100	100	100	100	100	66

Algorithme 3 – Marple – Intérêt

- ▶ est-ce que l'algorithme de Marple présente le même intérêt que celui de Levinson-Durbin ou de Burg en termes de haute-résolution ?
- ▶ on refait les mêmes tests qu'avec Levinson-Durbin
 - ▶ 2 sinusoïdes proches, etc. : voir transparent suivant
- ▶ et on fait varier l'ordre p
 - ▶ comme pour Levinson-Durbin et contrairement à Burg, il faut utiliser de grands ordres p
- ▶ le bruit blanc additif a pour variance $\sigma^2 = 1e^{-2}$

Algorithme 3 – Marple – Intérêt

- ▶ la longueur du signal est 1280 échantillons (une seule trame de 40 ms)
- ▶ sinusoïdes de fréquences 440 Hz et 466 Hz
- ▶ la DSP est calculée sur 32768 points
- ▶ méthode de détection simple : on détecte les maximums et minimums locaux entre $f_{min} - 40 \text{ Hz}$ et $f_{max} + 40 \text{ Hz}$
- ▶ puis on vérifie qu'il y a bien 2 maximums locaux, 1 minimum local, et que celui-ci se situe entre les 2 maximums locaux
- ▶ le résultat obtenu dépend de la phase respective des 2 sinus
- ▶ il dépend aussi de l'ordre du modèle AR (qu'il faut prendre grand)

Algorithme 3 – Marple – Intérêt

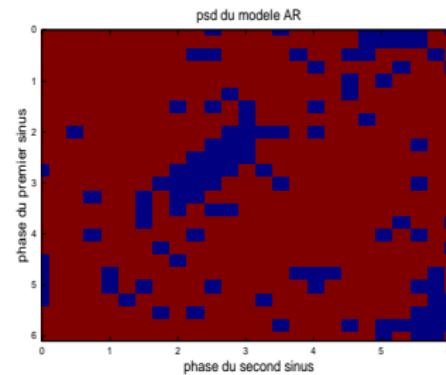
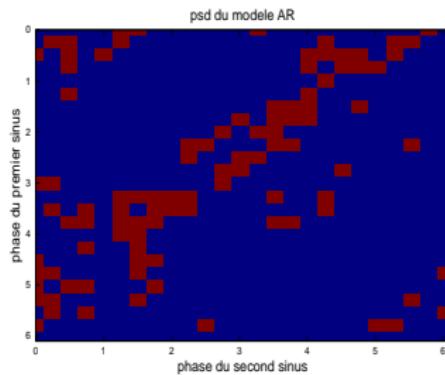
► voir :

scriptdoublonmarple.m

Algorithme 3 – Marple – Intérêt

Marple (ordre : 90) : 15.68 % de succès

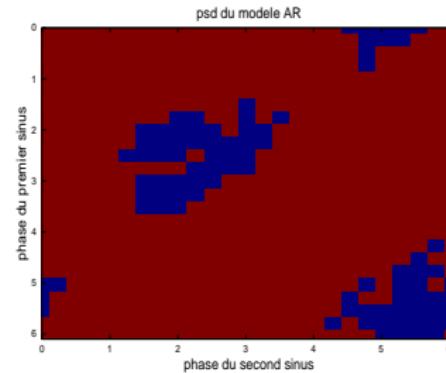
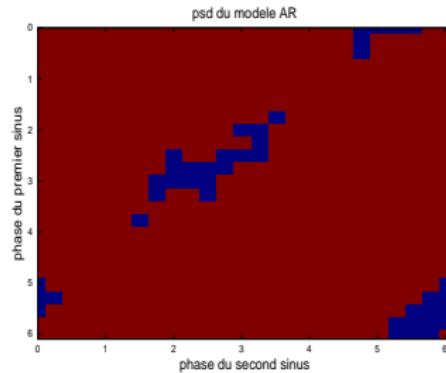
Marple (ordre : 100) : 82.72 % de succès



Algorithme 3 – Marple – Intérêt

Marple (ordre : 110) : 93.28 % de succès

Marple (ordre : 120) : 85.76 % de succès



Algorithme 3 – Marple – Intérêt

- ▶ là aussi, comme pour Levinson-Durbin et Burg, on fait mieux qu'avec la FFT
- ▶ la plage d'ordres p pour lesquels l'algorithme de Marple donne de bons résultats n'est pas très large non plus
- ▶ ça rend vraiment la mise en place de méthodes d'estimation de l'ordre p à utiliser nécessaire
- ▶ note 1 : bien sûr, il faudrait regarder de près les influences de TOL1 et TOL2
- ▶ note 2 : le bruit blanc additif a pour variance $\sigma^2 = 1e^{-2}$, ce qui est visible sur la forme d'onde

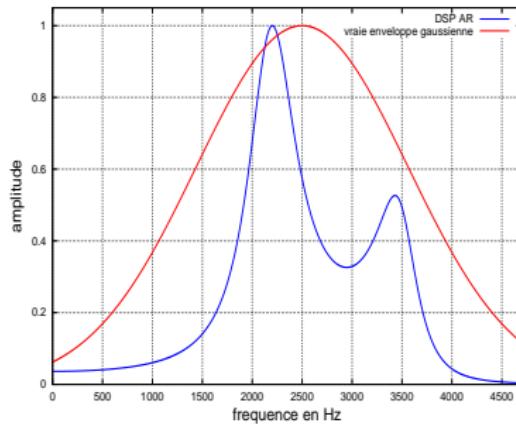
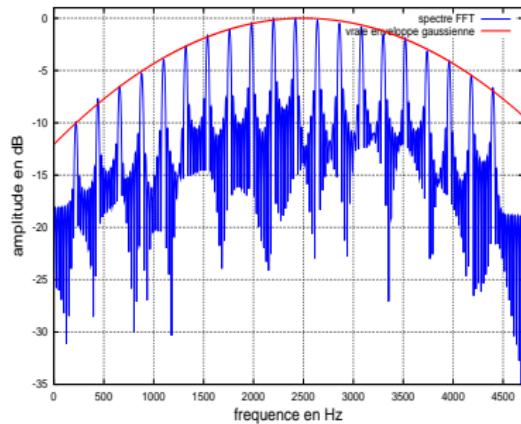
Algorithme 3 – Marple – Enveloppe

On a un signal harmonique (composé d'une somme de partiels harmoniques). Le signal considéré :

- ▶ fréquence fondamentale : $f_0 = 220 \text{ Hz}$
- ▶ nombre de partiels : 20
- ▶ amplitudes des partiels : elles suivent une gaussienne centrée en $f_{max} = 2500 \text{ Hz}$ et d'écart-type 1500
 - ▶ voir transparent suivant
- ▶ phases des partiels : tirées aléatoirement entre 0 et 2π
- ▶ la longueur du signal est toujours 1280 échantillons, et la fréquence d'échantillonnage 32000 Hz
- ▶ le bruit blanc additif a pour variance $\sigma^2 = 1e^{-2}$

Algorithme 3 – Marple – Enveloppe

Exemple de spectre FFT et d'enveloppe spectrale AR ($p = 10$, norme 1) obtenus avec le signal précédent :



Algorithme 3 – Marple – Enveloppe

Note : bien sûr, les pics de la FFT collent bien à la vraie enveloppe, mais par contre il faut encore détecter automatiquement ces pics, pour obtenir l'enveloppe spectrale !

Mesures :

- ▶ on calcule entre 0 Hz et 4700 Hz la différence pour chaque fréquence entre la DSP normalisée obtenue et la vraie enveloppe gaussienne \Rightarrow la somme normalisée donne la distance entre les deux
- ▶ on fait cette mesure un grand nombre de fois : le résultat dépend des phases respectives des partiels
- ▶ on fait cette mesure pour différents ordres p

Algorithme 3 – Marple – Enveloppe

► voir :

scriptenvelopmarple.m

Algorithme 3 – Marple – Enveloppe – Norme 1

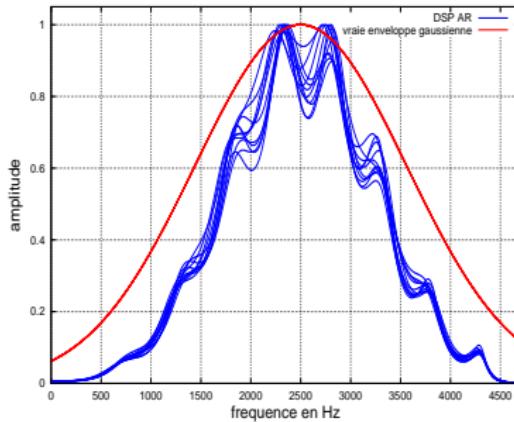
p	N	min	max	mean	σ
4	100	0.677920	0.724651	0.707395	0.007715
8	100	0.469369	0.530792	0.497521	0.010259
20	100	0.346454	0.454736	0.406071	0.021844
30	100	0.290828	0.427371	0.367696	0.028575
40	100	0.293570	0.470513	0.368925	0.034423
50	100	0.271987	0.418700	0.340990	0.027095
60	100	0.261239	0.404957	0.338043	0.027820
100	100	0.280259	0.421560	0.337116	0.025367
120	100	0.348928	0.535911	0.404892	0.028719

Algorithme 3 – Marple – Enveloppe

- ▶ il y a un souci avec la normalisation de la DSP AR, c'est-à-dire l'adaptation de son amplitude (voir le script ; voir aussi Levinson-Durbin)
- ▶ la normalisation utilisée pour obtenir les précédents résultats consistait seulement à mettre le maximum de la DSP AR à 1 (en pratique, par FFT, on pourrait estimer le maximum de l'enveloppe réelle, donc c'est d'accord) [norme 1]
- ▶ sur la figure suivante, on voit que ça amène à sous-estimer l'amplitude de l'enveloppe
- ▶ on pourrait normaliser la surface de la DSP AR entre 0 Hz et 4700 Hz pour qu'elle corresponde à celle de la vraie DSP (en pratique, on n'a pas facilement accès à cette information) [norme 2]

Algorithme 3 – Marple – Enveloppe – Norme 1

$p = 50$:



Algorithme 3 – Marple – Enveloppe – Norme 2

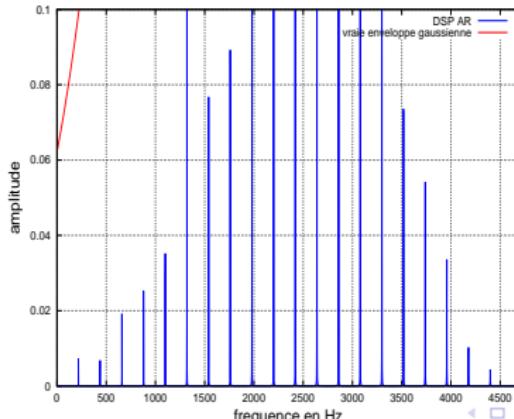
Utiliser la deuxième normalisation permet de réduire notablement l'erreur :

p	N	min	max	mean	σ
4	100	0.696544	0.748435	0.725153	0.011198
8	100	0.426334	0.497082	0.465793	0.012613
20	100	0.305580	0.369702	0.339321	0.011983
30	100	0.301040	0.341501	0.320478	0.008217
40	100	0.292562	0.333391	0.315934	0.008176
50	100	0.301565	0.332843	0.316036	0.006306
60	100	0.294514	0.329529	0.309922	0.007486
100	100	0.297681	0.328338	0.310193	0.005744
120	100	0.315132	0.363284	0.330026	0.009410

Algorithme 3 – Marple – Enveloppe – Norme 1

Est-ce qu'en augmentant suffisamment l'ordre p on arrive à récupérer les 20 partiels ?

Oui ! Ainsi, pour $p = 140$, on obtient par exemple, en zoomant sur y :



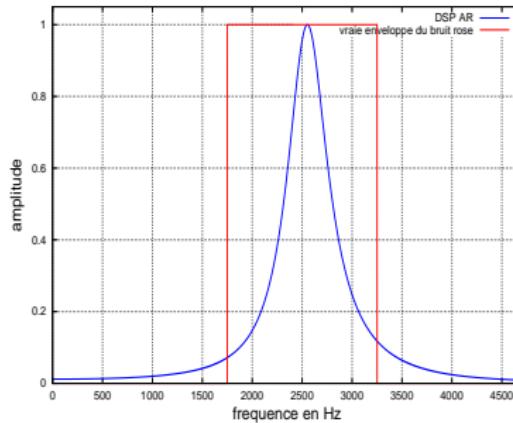
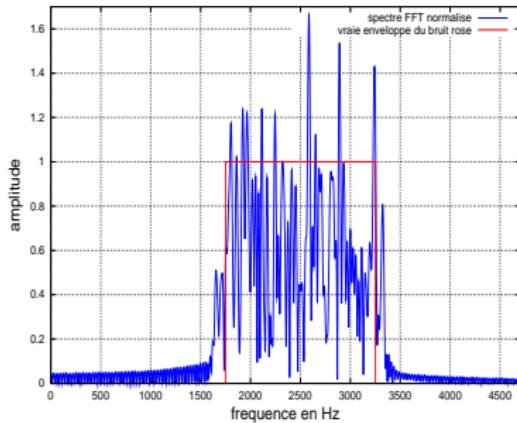
Algorithme 3 – Marple – Bruit rose

Le bruit rose considéré :

- ▶ fréquence centrale du bruit rose : $f_c = 2500 \text{ Hz}$
- ▶ largeur de la bande : 1500 Hz
- ▶ l'amplitude du bruit vaut 1 dans la bande, et 0 donc en-dehors
 - ▶ voir transparent suivant
- ▶ on fait d'abord un bruit blanc, qu'on filtre après coup
- ▶ la longueur du signal est toujours 1280 échantillons, et la fréquence d'échantillonnage 32000 Hz

Algorithme 3 – Marple – Bruit rose

Exemple de spectre FFT et d'enveloppe spectrale AR ($p = 10$, norme 1) obtenus avec le signal précédent :



Algorithme 3 – Marple – Bruit rose

Note : la normalisation du spectre FFT est parfaitement contrôlée
(considérations énergétiques concernant les spectres continus)

Mesures :

- ▶ on calcule entre 0 Hz et 4700 Hz la différence pour chaque fréquence entre la DSP normalisée obtenue et la vraie enveloppe du bruit rose \Rightarrow la somme normalisée donne la distance entre les deux
- ▶ on fait cette mesure un grand nombre de fois : le résultat dépend du tirage initial du bruit blanc
- ▶ on fait cette mesure pour différents ordres p

Algorithme 3 – Marple – Bruit rose

► voir :

scriptpinkmarple.m

Algorithme 3 – Marple – Bruit rose – Norme 1

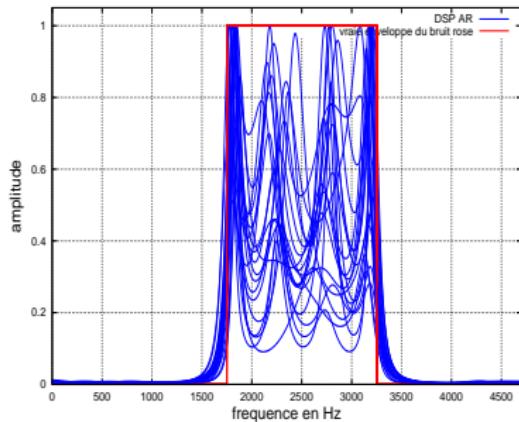
p	N	min	max	mean	σ
4	100	0.632735	0.710109	0.670964	0.014961
8	100	0.673585	0.769351	0.722897	0.018328
20	100	0.392370	0.691596	0.549963	0.069676
30	100	0.349443	0.737357	0.554651	0.090075
40	100	0.314596	0.779568	0.519644	0.099164
50	100	0.411443	0.805521	0.572960	0.089711
60	100	0.356923	0.811290	0.578794	0.096711
100	100	0.467462	0.884309	0.648232	0.090634
120	100	0.468388	0.933699	0.673585	0.098962

Algorithme 3 – Marple – Bruit rose

- ▶ il y a un souci avec la normalisation de la DSP AR (contrairement à ce qui se passe avec Levinson-Durbin et Burg)
- ▶ la normalisation utilisée pour obtenir les précédents résultats consistait seulement à mettre le maximum de la DSP AR à 1 (**en pratique, par FFT, on a un peu de mal à estimer le maximum de l'enveloppe réelle**) [norme 1]
- ▶ sur la figure suivante, on voit que ça amène comme précédemment à sous-estimer l'amplitude de l'enveloppe
- ▶ on pourrait normaliser la surface de la DSP AR entre 0 *Hz* et 4700 *Hz* pour qu'elle corresponde à celle de la vraie DSP (**en pratique, on n'a pas facilement accès à cette information**) [norme 2]

Algorithme 3 – Marple – Bruit rose – Norme 1

$p = 50$:



Algorithme 3 – Marple – Bruit rose – Norme 2

Utiliser la deuxième normalisation permet de réduire notablement l'erreur :

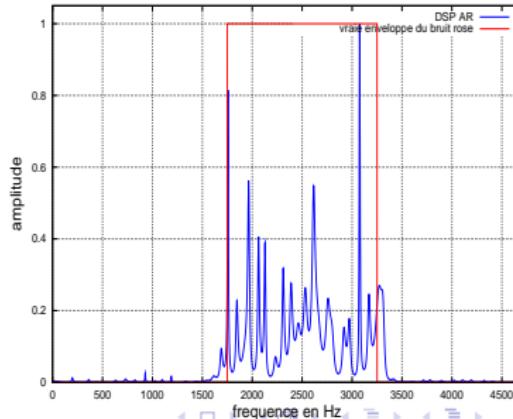
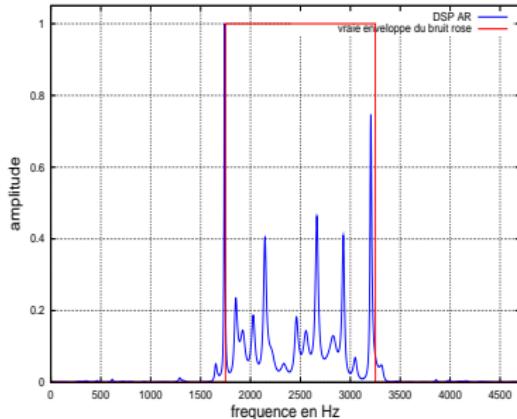
p	N	min	max	mean	σ
4	100	0.651150	0.783264	0.712398	0.026710
8	100	0.730459	0.925625	0.812793	0.040681
20	100	0.273703	0.714369	0.473830	0.076885
30	100	0.276963	0.557710	0.391849	0.064405
40	100	0.216831	0.513790	0.343504	0.065701
50	100	0.232368	0.611272	0.393878	0.084609
60	100	0.249857	0.574095	0.403160	0.074554
100	100	0.273976	0.699720	0.450046	0.077549
120	100	0.334253	0.658589	0.478208	0.069821



Algorithme 3 – Marple – Bruit rose – Norme 1

Et si on prend un ordre p extrêmement grand, qu'est-ce qui se passe ?

Pour $p = 300$ et $p = 400$, on obtient par exemple :



Algorithmes – Résumé des résultats

	Levinson-Durbin	Burg	Marple	FFT
1 sinus	$p \geq 4$	$p \geq 2$	$3 \leq p \leq 10$ $\sigma^2 = 1e^{-4}$ $4 \leq p \leq 40$ $\sigma^2 = 1e^{-3}$	oui
2 sinus	$p = 250 : 63\%$ $p = 265 : 88\%$ $p = 300 : 100\%$	$p = 4 : 99.68\%$ $p = 5 : 100\%$ $p = 6 : 97\%$ $p = 7 : 7\%$	$p = 90 : 16\%$ $p = 100 : 83\%$ $p = 110 : 93\%$ $p = 120 : 86\%$	Blackman 32 % Hanning 40 % (Rect. 64 % méth. autre)
Enveloppe	<i>norm1</i> : $e > 0.318$ <i>norm2</i> : $e > 0.305$	pas très efficace	<i>norm1</i> : $e > 0.337$ <i>norm2</i> : $e > 0.310$	détection des pics à faire
20 partiels	oui ($p = 200$)	oui ($p = 100$)	oui ($p = 140$)	oui
bruit rose	<i>norm1</i> : $e > 0.568$ <i>norm2</i> : $e > 0.367$ <i>norm3</i> : $e > 0.378$	<i>norm1</i> : $e > 0.708$ <i>norm2</i> : $e > 0.456$ <i>norm3</i> : $e > 0.493$	<i>norm1</i> : $e > 0.520$ <i>norm2</i> : $e > 0.344$	$e \simeq 0.5$

Algorithme 3 – Marple – Les amplitudes (norme 3)

- ▶ *mymarple_matlab.m* : l'amplitude du sinus est 1, et on obtient un pic gigantesque (amplitude de l'ordre de 400000)
- ▶ *scriptdoublonmarple.m* : l'amplitude de chaque sinus vaut 1, et on obtient des pics gigantesques (amplitudes quasi égales, et de l'ordre de $1e^6$)
- ▶ *scriptenvelopmarple.m* : l'amplitude du plus grand partiel est 1, et on obtient un maximum gigantesque (de l'ordre de 100000)
- ▶ *scriptpinkmarple.m* : l'amplitude dans la bande du bruit rose est 1, et on obtient une DSP d'amplitude trop grande (de l'ordre de 2000)

Algorithme 3 – Marple – Le code matlab/octave

- ▶ j'ai mis le code de *mymarple_matlab.m* sur mon site :
[http://www.metz.supelec.fr/metz/personnel/
rossignol/mymarple_matlab.m](http://www.metz.supelec.fr/metz/personnel/rossignol/mymarple_matlab.m)
- ▶ vous pourriez aussi essayer de trouver des solutions personnelles aux divers problèmes rencontrés

Algorithme 3 – Marple – Le code en C

- ▶ dans une perspective d'utilisation de ces méthodes d'analyse spectrale en temps réel par exemple, il faudrait traduire ce code en C (par exemple)
- ▶ si on examine le code matlab/octave, on voit qu'il n'y a rien de compliqué
- ▶ donc, pas besoin a priori d'utiliser de bibliothèques pré-existantes par exemple pour le calcul matriciel, etc.
- ▶ bien sûr, il faudrait aussi considérer tous les problèmes liés à l'acquisition des signaux

Téléchargements

- ▶ cours sur les Modèles de perception et de production :
[http://www.metz.supelec.fr/metz/personnel/rossignol/
slides_modeleperception.pdf](http://www.metz.supelec.fr/metz/personnel/rossignol/slides_modeleperception.pdf)
- ▶ http://www.metz.supelec.fr/metz/personnel/rossignol/slides_analyse.pdf
- ▶ [http://www.metz.supelec.fr/metz/personnel/rossignol/
mylevinsondurbin.m](http://www.metz.supelec.fr/metz/personnel/rossignol/mylevinsondurbin.m)
- ▶ [http://www.metz.supelec.fr/metz/personnel/rossignol/
scriptphaselevinson.m](http://www.metz.supelec.fr/metz/personnel/rossignol/scriptphaselevinson.m)
- ▶ <http://www.metz.supelec.fr/metz/personnel/rossignol/myburg.m>
- ▶ http://www.metz.supelec.fr/metz/personnel/rossignol/mymarple_matlab.m
- ▶ <http://www.metz.supelec.fr/metz/personnel/rossignol/mypisarenko.m>
- ▶ http://www.metz.supelec.fr/metz/personnel/rossignol/mymusic_matlab.m ↗ ↘ ↙

Différences essentielles entre les divers algorithmes

- ▶ **Levinson-Durbin** : inversion itérative de la matrice d'autocorrélation
- ▶ **Burg** : minimisation de l'erreur de prédiction (avant + arrière) par rapport au coefficient de réflexion obtenu à chaque itération :

$$\frac{\partial E_p}{\partial a_{k,k}} = 0$$

- ▶ **Marple** : minimisation de l'erreur de prédiction (avant + arrière) par rapport à tous les paramètres AR obtenus à chaque itération :

$$\frac{\partial E_p}{\partial a_{k,i}} = 0 \quad \forall i$$

- ▶ **Morf** : minimisation de l'erreur de prédiction à chaque itération, mais cette fois on considère que les coefficients AR ne sont pas les mêmes pour la prédiction avant et pour la prédiction arrière
- ▶ et la liste pourrait continuer, longue...

Quelques points saillants (négatifs) des 3 méthodes testées

- ▶ **Levinson-Durbin**
 - ▶ estimation des coefficients d'autocorrélation (imparfaite et possiblement longue)
 - ▶ résolution pas forcément extraordinaire (à tester plus)
- ▶ **Burg**
 - ▶ dédoublement possible des raies, quand le rapport signal sur bruit (SNR) est fort notamment (à tester plus)
 - ▶ estimation des positions des pics dépendant de la phase, c'est-à-dire biais (à tester plus)
- ▶ **Marple** : stabilité incertaine (ça, on s'en rend compte aisément)
- ▶ **Toutes :**
 - ▶ comment choisir l'ordre ?
 - ▶ résolution décroissante avec le SNR (à tester)
 - ▶ amplitudes des raies non liées à celles des sinusoïdes

Quelques points positifs de ces méthodes AR

- ▶ En terme de résolution, Marple a montré que 2 sinusoïdes noyées dans du bruit pouvaient être séparées pour des Δf de cet ordre :

$$\Delta f \simeq \frac{1.03}{p [(p+1)\text{SNR}]^{0.31}} \text{ pour les modèles AR}$$

$$\Delta f \simeq \frac{0.86}{N} \text{ pour le périodogramme}$$

- ▶ algorithmes simples et rapides (? !)
- ▶ résultats pas pires que ceux des méthodes non-paramétriques
- ▶ résultats exploitables !

Les modèles AR avec les mains

- ▶ on va faire le lien entre :
 - ▶ la position des zéros de $A(z)$ avec (slide 18) :
$$A(z) = 1 + a_1z^{-1} + a_2z^{-2} + \dots$$
 - ▶ les coefficients AR : $\{1, a_1, a_2, \dots\}$
 - ▶ et la DSP obtenue : $P_{xx}(\nu) = \frac{1}{\left|1 + \sum_{k=1}^p a_k \exp(-j2\pi k\nu)\right|^2}$
- ▶ on va jouer avec *aveclesmains.m*

Les modèles AR avec les mains

► essai 1 :

- ▶ on a 2 pôles complexes conjugués \Rightarrow donc le signal est réel
- ▶ de phase constante égale $\pm 36^\circ$; ça correspond donc à une raie de fréquence réduite $\nu = 36/360 = 0.1$ ou de fréquence $f = 3200 \text{ Hz}$ si $f_e = 32000 \text{ Hz}$
- ▶ et on augmente leur amplitude progressivement

► essai 2 :

- ▶ on a 2 pôles complexes conjugués \Rightarrow donc le signal est réel
- ▶ de phase constante égale $\pm 90^\circ$; ça correspond donc à une raie de fréquence réduite $\nu = 90/360 = 0.25$ ou de fréquence $f = 8000 \text{ Hz}$ si $f_e = 32000 \text{ Hz}$
- ▶ et on augmente leur amplitude progressivement

Les modèles AR avec les mains

► essai 3 :

- ▶ on a 2 pôles complexes conjugués \Rightarrow donc le signal est réel
- ▶ leur amplitude est constante : 0.9
- ▶ leur phase varie de 10° à 170° (pour les pôles de partie imaginaire positive)

► essai 4 : on a 6 pôles :

- ▶ $0.99 \exp(\pm j2\pi 0.05) \quad 0.98 \exp(\pm j2\pi 0.20) \quad 0.8 \exp(\pm j2\pi 0.3)$
- ▶ les pôles complexes sont conjugués 2 à 2 \Rightarrow donc le signal est réel

On en a fini pour le moment avec les divers algorithmes de l'autorégression !

Page intentionnellement laissée blanche

Et on entre dans le monde de la décomposition de la matrice de corrélation en sous-espace bruit et en sous-espace signal !

Page intentionnellement laissée blanche

- ▶ Introduction
- ▶ Modèles AR
- ▶ Pisarenko

Pisarenko

- ▶ on ne considère plus le même modèle de signal que pour les modèles AR/MA/ARMA
- ▶ on fait l'hypothèse que le signal est la somme de m raies entachées d'un bruit blanc gaussien :

$$x(n) = \sum_{i=1}^m a_i \cos\left(2\pi f_i \frac{n}{f_e} + \phi_i\right) + n(n)$$

- ▶ on veut trouver les amplitudes a_i , les fréquences f_i et la variance du bruit σ^2
- ▶ note : on laisse tomber les phases ϕ_i (de toute façon, c'est cohérent avec notre perspective de traitement des sons, puisque notre oreille n'est pas sensible aux phases)

Pisarenko

- ▶ **point 1.** les coefficients d'autocorrélation théoriques d'un tel signal sont (se rappeler le TL de Sig1) :

$$R_{xx}(k) = \frac{1}{2} \sum_{i=1}^m a_i^2 \cos\left(2\pi f_i \frac{k}{f_e}\right) + \sigma^2 \delta(k)$$

- ▶ ceci permet de remonter aux amplitudes des raies une fois qu'on a estimé leurs fréquences (voir le point 2. plus loin)
- ▶ on a en effet (**P4**) :

$$\begin{bmatrix} \cos(2\pi f_1/f_e) & \dots & \cos(2\pi f_m/f_e) \\ \vdots & \ddots & \vdots \\ \cos(2\pi mf_1/f_e) & \dots & \cos(2\pi mf_m/f_e) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_m \end{bmatrix} = \begin{bmatrix} \hat{R}_{xx}(1) \\ \vdots \\ \hat{R}_{xx}(m) \end{bmatrix}$$

Pisarenko

- ▶ il faut maintenant inverser la matrice des cos !
- ▶ note : et ici on oublie tranquillement le bruit
- ▶ à partir de m coefficients d'autocorrélation estimés et des f_m , après avoir inversé la matrice des cos, on obtient facilement les α
- ▶ et bien sûr les amplitudes des raies sont alors :

$$a_i = \sqrt{|2\alpha_i|}$$

- ▶ note : les phases disparaissent dès lors qu'on se limite aux coefficients d'autocorrélation

Pisarenko

- ▶ **point 2.** il s'agit de trouver les fréquences f_m
- ▶ il faut maintenant remarquer que, dans le cas simple où il n'y a qu'une seule sinusoïde de fréquence f , on a :

$$x(n) = 2 \cos(2\pi f/f_e) x(n-1) - x(n-2)$$

- ▶ note : il suffit de se rappeler que :

$$\cos(a)\cos(b) = \frac{1}{2}(\cos(a-b) + \cos(a+b))$$

- ▶ ce résultat peut se généraliser pour une somme de m sinusoïdes, et l'on peut écrire :

$$x(n) = \sum_{i=1}^{2m} a_i x(n-i)$$

Pisarenko

- ▶ de plus, il peut se généraliser à la version bruitée du modèle de Pisarenko ; on a alors :

$$x(n) - n(n) = \sum_{i=1}^{2m} a_i (x(n-i) - n(n-i))$$

ou

$$\sum_{i=0}^{2m} a_i x(n-i) = \sum_{i=0}^{2m} a_i n(n-i) \quad (\text{avec } a_0 = 1) \quad \text{P1}$$

- ▶ c'est un signal ARMA, mais on s'en moque pour la suite : comme la partie AR et la partie MA sont identiques, la technique utilisée pour trouver les paramètres est complètement différente de celles utilisées pour les ARMA (ouf : on ne retombe pas dans l'autorégressif !)

Pisarenko

- ▶ on a besoin de se souvenir de quelques éléments d'algèbre linéaire :
 - ▶ comme le signal est réel, les racines de $A(z)$ vont par paires conjuguées
 - ▶ comme le signal est une somme de raies, les racines de ce polynôme sont sur le cercle unité
 - ▶ et un polynôme dont les racines vont par paires conjuguées de module 1 est symétrique
- ▶ on a donc : $a_i = a_{2m-i}$ $i \in [0, m]$

Pisarenko

- ▶ on peut écrire **P1** sous forme matricielle : $\mathbf{x}^T \mathbf{a} = \mathbf{n}^T \mathbf{a}$
- ▶ on a donc aussi : $\mathbf{x}\mathbf{x}^T \mathbf{a} = \mathbf{x}\mathbf{n}^T \mathbf{a}$
- ▶ or la matrice d'autocorrélation est l'espérance de $\mathbf{x}\mathbf{x}^T$:
 $\hat{R}_{xx} = E[\mathbf{x}\mathbf{x}^T]$, qui est donc de taille $(2m+1) \times (2m+1)$
- ▶ de plus, comme le bruit est supposé être décorrélé de la somme de sinusoïdes (notée ici \mathbf{s}), on a :

$$E[\mathbf{x}\mathbf{n}^T] = E[(\mathbf{s} + \mathbf{n})\mathbf{n}^T] = E[\mathbf{n}\mathbf{n}^T] = \sigma^2 \mathbf{Id}$$

- ▶ ce qui nous fait aboutir à : $\hat{R}_{xx}\mathbf{a} = \sigma^2 \mathbf{a}$
- ▶ donc le vecteur des paramètres \mathbf{a} est le vecteur propre de \hat{R}_{xx} associé à la valeur propre σ^2 (toujours de l'algèbre)

Pisarenko

- ▶ il faut voir que cette valeur propre est en fait la plus petite valeur propre de \hat{R}_{xx}
- ▶ en effet, on peut voir que : $\hat{R}_{xx} = R_{ss} + R_{nn}$, ou encore $R_{ss} = \hat{R}_{xx} - \sigma^2 \mathbf{Id}$
- ▶ or, si on note λ_i les autres valeurs propres de \hat{R}_{xx} , les valeurs propres de R_{ss} sont 0 et $\lambda_i - \sigma^2$
- ▶ et comme (autre rappel d'algèbre linéaire) R_{ss} est définie positive, puisque c'est une matrice d'autocorrélation, alors toutes ses valeurs propres sont positives ; on a donc :
$$\sigma^2 < \lambda_i \quad \forall i$$
, et la démonstration est finie

Pisarenko

- ▶ **point 3.** on ne cherche pas toutes les valeurs propres de \hat{R}_{xx}
- ▶ il y a un algorithme récursif qui permet de trouver la plus petite, ainsi que son vecteur propre associé
- ▶ c'est l'algorithme de la puissance inverse ; à l'itération k , on a :

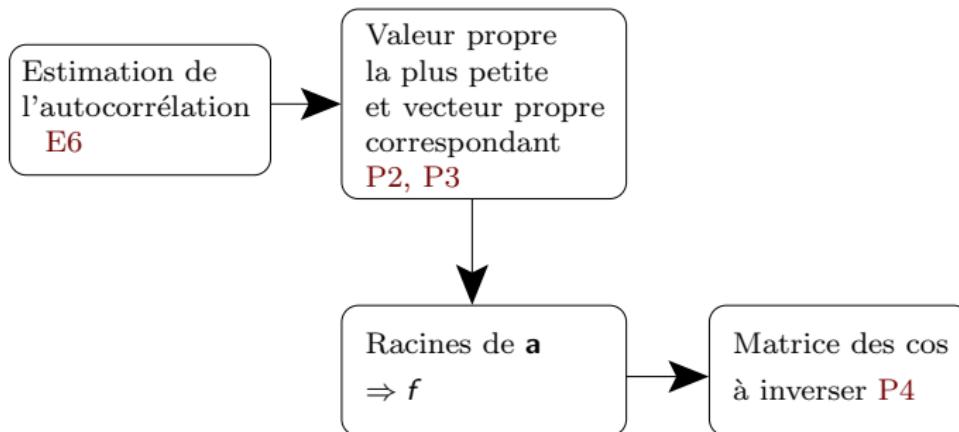
$$\mu_k = \frac{\mathbf{a}_k^T \hat{R}_{xx} \mathbf{a}_k}{\mathbf{a}_k^T \mathbf{a}_k} \quad \text{et} \quad \mathbf{a}_{k+1} = \hat{R}_{xx}^{-1} \mu_k \mathbf{a}_k \quad \text{P2 et P3}$$

- ▶ la suite μ_k converge vers la plus petite valeur propre de \hat{R}_{xx} et la suite des vecteurs \mathbf{a}_k vers le vecteur propre associé

Pisarenko

- ▶ note 1 : on initialise μ_0 à 1 et \mathbf{a}_0 aléatoirement
- ▶ note 2 : de plus, il faut un critère d'arrêt pour l'algorithme de la puissance inverse :
 - ▶ quand la différence entre μ_{k+1} et μ_k est inférieure à un certain seuil ($1e^{-10}$ pour nous ici), on décide qu'on a convergé
 - ▶ les racines du polynôme associé à \mathbf{a} sont les $\exp(\pm j2\pi f_i/f_e)$ qui permettent de remonter aux f_i
 - ▶ et on a tout ce qu'il nous faut

Pisarenko – Résumé



Pisarenko – Code matlab

```
function [ff, mydsp] = mypisarenko(xx, pp, fe)
MM1 = 2*pp ;MM=MM1+1;NN=length(xx);xx = xx-mean(xx); %% initialisations
acf = xcorr(xx(1 :NN), MM1, 'biased');IMM=length(acf); %% corrélation
rrr1 = acf(MM1+1 :IMM)';
for ii=1 :MM1    rrr1 = [rrr1 acf(MM1+1-ii :IMM-ii)'];    end; %% matrice de corrélation
rrr1 = rrr1';rrr = rrr1;irrr=inv(rrr);
mu=1; %% récursion pour trouver la plus petite valeur propre
muold=mu;
aaa=rand(1,2*pp+1);
kk=1;
while kk==1
    mu = aaa*rrr*aaa'/(aaa*aaa');
    aaa = irrr*mu*aaa';aaa = aaa';
    if (abs(muold-mu)<1e-10)      kk=0;    end; %% arrêt
    muold=mu;
end;
racines = roots(aaa);frequ = log(racines)/j/2/pi*fe;frequ = abs(frequ(1 :2 :end)); %% extract. racines ⇒ raies
df=0.9765625;ff=fe/2 :df :fe/2 ;mydsp=zeros(length(ff),1)+1e-10; %% la dsp est calculée tous les df Hz
for ii=1 :pp      for jj=1 :pp      coscos(ii,jj) = cos(2.*pi*ii*frequ(jj)/fe);    end;    end;
alpha = inv(coscos)*acf(2*pp+1 :3*pp)';amp = sqrt(2*abs(alpha)); %% amplitudes
for ii=1 :pp %% densité spectrale de puissance (c'est un spectre de raies)
    [mini, posi] = min(abs(ff-frequ(ii)));mydsp(posi) = amp(ii); %% fréquences positives
    [mini, posi] = min(abs(ff+frequ(ii)));mydsp(posi) = amp(ii); %% fréquences négatives
end;
```

Pisarenko – Premiers tests

On va faire quelques tests, avec des signaux (sons) simples pour vérifier qu'on obtient bien des densités spectrales de puissance intéressantes. Les 4 tests qu'on considère ici sont :

- ▶ on a une sinusoïde pure \Rightarrow est-ce qu'on arrive à la mettre en évidence ?
- ▶ on a deux sinusoïdes proches en fréquence \Rightarrow est-ce qu'on arrive à les discriminer ?
- ▶ on a un signal compliqué : une somme de sinusoïdes harmoniques \Rightarrow est-ce qu'on arrive à obtenir une enveloppe spectrale, modélisant ce signal, et/ou les n partiels ?
- ▶ on a un signal compliqué : un bruit rose \Rightarrow est-ce qu'on arrive à obtenir une enveloppe spectrale, modélisant ce signal ?

Pisarenko – Une sinusoïde

mypisarenko.m

Paramètres de l'exemple :

- ▶ $f_e = 32000$ Hz
- ▶ signal : une sinusoïde présente
 - ▶ de fréquence 440 Hz
 - ▶ et de phase aléatoire, tirée entre 0 et 2π
 - ▶ de taille 1280 échantillons (40 ms)
- ▶ $p = 1$
- ▶ **attention** : la méthode historique permet de récupérer un spectre de raies, pas une DSP complète ; on a :
 - ▶ ou un spectre de raies (on met ϵ aux autres fréquences)
 - ▶ ou une DSP, en faisant comme pour MUSIC (voir plus loin)

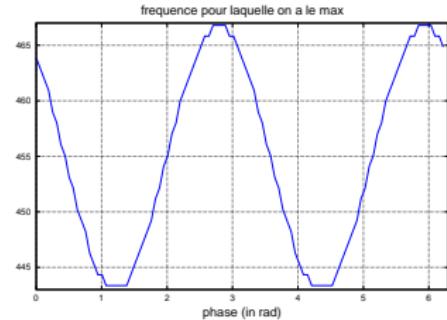
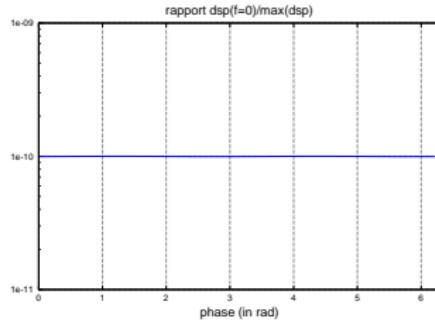
Pisarenko – Une sinusoïde

scriptphasepisarenko.m

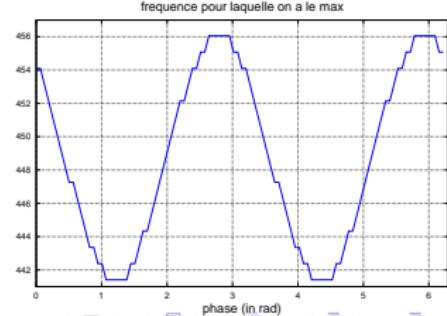
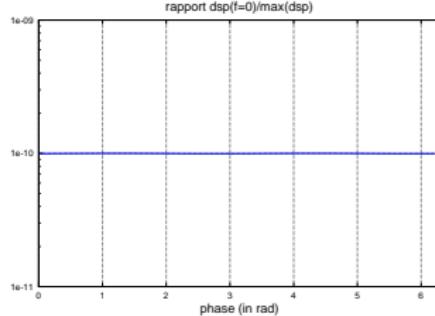
- ▶ on regarde ce qui se passe en fonction de la phase de la sinusoïde
- ▶ on mesure le rapport entre l'amplitude du spectre de raies en $f = 0$ et son amplitude maximum (sensée se trouver à $\pm 440 \text{ Hz}$)
- ▶ on regarde où se trouve en réalité l'amplitude maximum du spectre de raies
- ▶ ce pour différents ordres : $p = 1, p = 2, p = 3, \dots$

Pisarenko – Une sinusoïde

$p = 1 :$



$p = 2 :$



Pisarenko – Une sinusoïde

- ▶ On donne le nombre de fois (sur 100) où la fréquence du maximum a été trouvée entre 400 Hz et 480 Hz.

p	1	2	3	10	100
%	100	100	100	100	100

- ▶ Malgré ce tableau prometteur, un ordre trop élevé devient un peu problématique dès 2
 - ▶ en effet, pour $p = 2$, l'amplitude du pic supplémentaire est 10 fois plus petite que celle du pic autour de 440 Hz
 - ▶ ça correspond à -20 dB (à comparer au -13 dB du 1er lobe secondaire pour le fenêtre rectangulaire... et aux -58 dB pour la fenêtre de Blackman)
 - ▶ bien sûr, la méthode est faite pour trouver une raie (dans les fréquences positives : on a la symétrie pour les fréquences négatives) pour chaque unité d'ordre, donc la méthode le fait

Pisarenko – 2 sinusoïdes

- ▶ est-ce que l'algorithme de Pisarenko présente un intérêt en termes de haute-résolution ?
- ▶ ON SE REND VITE COMPTE QUE ÇA NE VA PAS ÊTRE TRÈS AISÉMENT LE CAS
 - ▶ on a 2 sinusoïdes, de fréquences $f_0 = 440 \text{ Hz}$ et $f_1 = 440 + f \text{ Hz}$
 - ▶ on diminue progressivement f et on mesure la capacité de la méthode à discriminer les 2 sinusoïdes
 - ▶ pour la méthode de mesure, voir le transparent suivant
- ▶ on fixe l'ordre p à 2

Pisarenko – 2 sinusoïdes

- ▶ longueur du signal : 1280 échantillons (une seule trame de 40 ms)
- ▶ spectre de raies calculé sur 32768 points
- ▶ méthode de détection simple : on détecte les maximums et minimums locaux entre $f_{min} - 40 \text{ Hz}$ et $f_{max} + 40 \text{ Hz}$
- ▶ puis on vérifie qu'il y a bien 2 maximums locaux, 1 minimum local, et que celui-ci se situe entre les 2 maximums locaux
- ▶ le résultat obtenu dépend de la phase respective des 2 sinus
- ▶ IL DÉPEND AUSSI DE LA DISTANCE f EN Hz ENTRE LES DEUX SINUSOÏDES

Pisarenko – 2 sinusoïdes

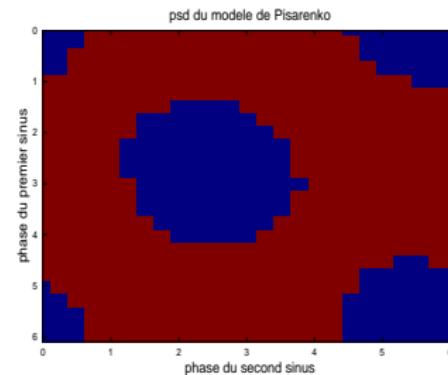
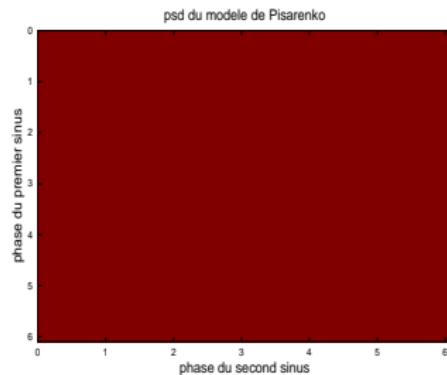
► voir :

scriptdoublonpisarenko.m

Pisarenko – 2 sinusoïdes

Pisarenko ($f = 4000 \text{ Hz}$) : 100 % de succès

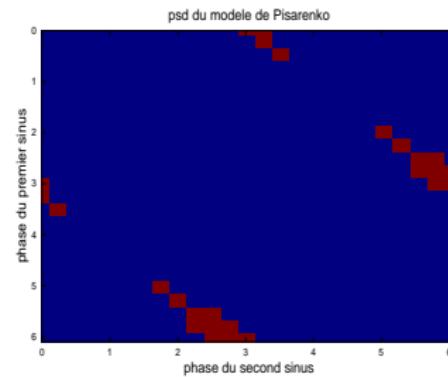
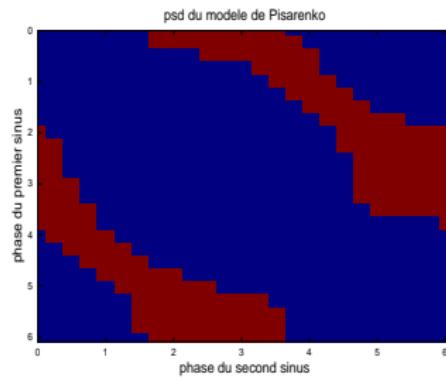
Pisarenko ($f = 3000 \text{ Hz}$) : 71.20 % de succès



Pisarenko – 2 sinusoïdes

Pisarenko ($f = 2000 \text{ Hz}$) : 26.72 % de succès

Pisarenko ($f = 1000 \text{ Hz}$) : 4.16 % de succès



Pisarenko – 2 sinusoïdes

- ▶ et si on prend une ordre p très grand ?
- ▶ il faut changer de méthode de détection, à cause des pics parasites
 - ▶ longueur du signal : 1280 échantillons (une seule trame de 40 ms)
 - ▶ sinusoïdes de fréquences 440 Hz et 466 Hz
 - ▶ spectre de raies calculé sur 32768 points
 - ▶ on détecte les deux pics de plus grandes amplitudes (dans les fréquences positives)
 - ▶ puis on regarde s'ils sont dans la bande de 400 Hz à 506 Hz
 - ▶ le résultat obtenu dépend de la phase respective des 2 sinus

Pisarenko – 2 sinusoïdes

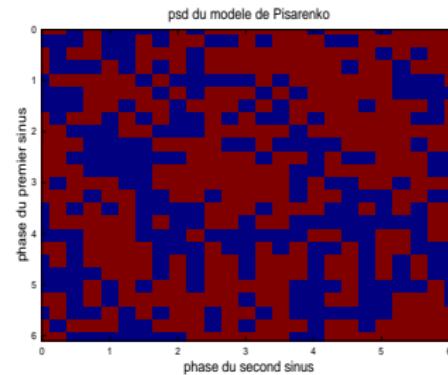
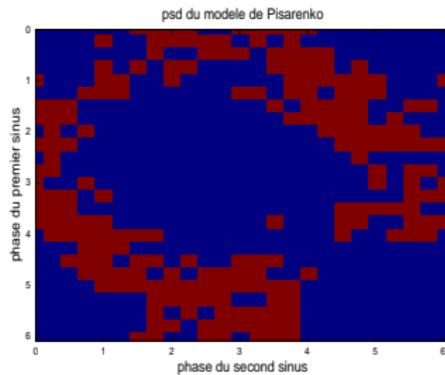
► voir :

scriptdoublonpisarenko2.m

Pisarenko – 2 sinusoïdes

Pisarenko ($p = 40$) : 32.64 % de succès

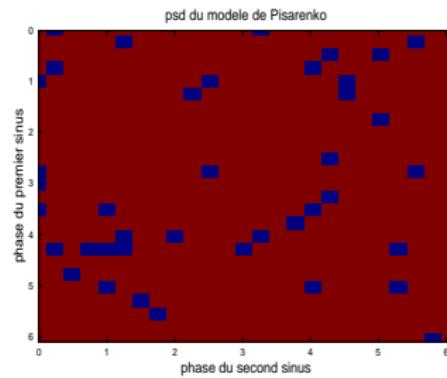
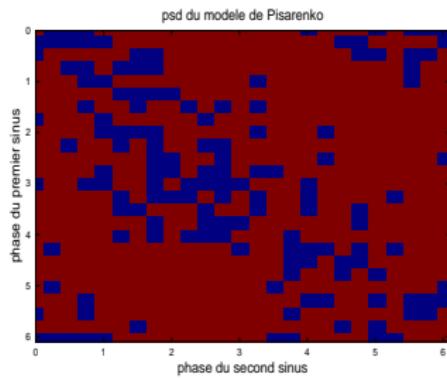
Pisarenko ($p = 60$) : 58.88 % de succès



Pisarenko – 2 sinusoïdes

Pisarenko ($p = 100$) : 77.12 % de succès

Pisarenko ($p = 150$) : 93.60 % de succès



Pisarenko – Enveloppe

- ▶ on considère toujours le spectre de raies
- ▶ bien sûr, on ne s'attend pas à obtenir quelque chose de fantastique (puisque la méthode donne des raies)
- ▶ on a un signal harmonique (composé d'une somme de partiels harmoniques). Le signal considéré :
 - ▶ fréquence fondamentale : $f_0 = 220 \text{ Hz}$
 - ▶ nombre de partiels : 20
 - ▶ amplitudes des partiels : elles suivent une gaussienne centrée en $f_{max} = 2500 \text{ Hz}$ et d'écart-type 1500
 - ▶ voir transparent suivant
 - ▶ phases des partiels : tirées aléatoirement entre 0 et 2π
 - ▶ la longueur du signal est toujours 1280 échantillons, et la fréquence d'échantillonnage 32000 Hz

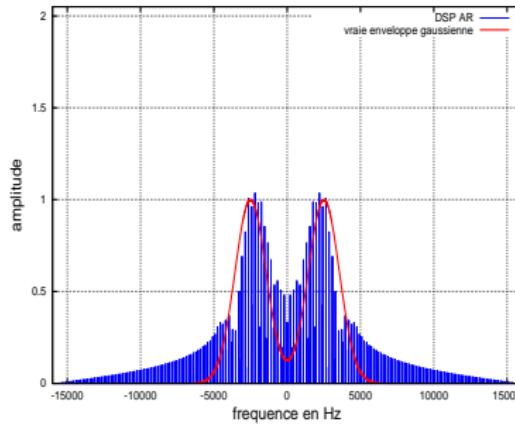
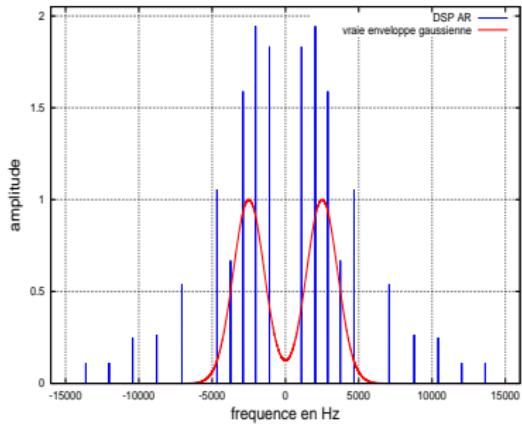
Pisarenko – Enveloppe

► voir :

scriptenveloppisarenko.m

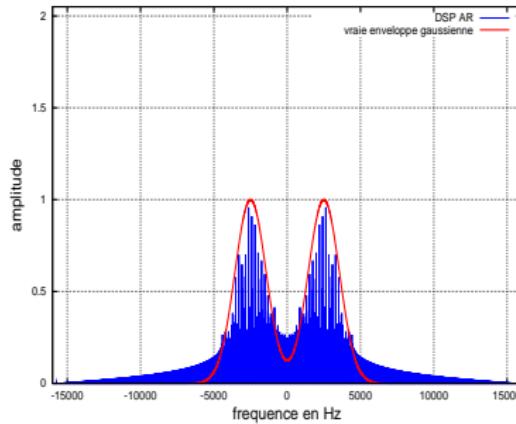
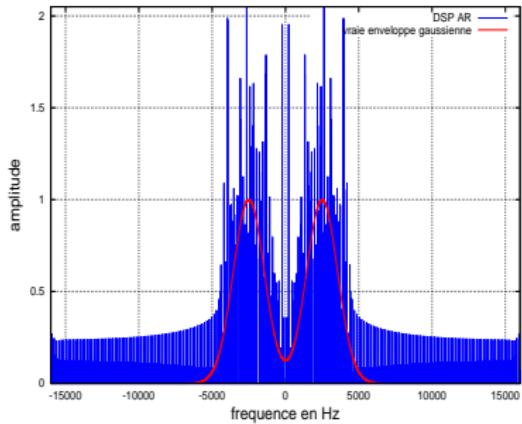
Pisarenko – Enveloppe

Spectres obtenus avec le signal précédent, $p = 10$ et $p = 100$, norme 3 :



Pisarenko – Enveloppe

Spectres obtenus avec le signal précédent, $p = 200$ et $p = 300$, norme 3 :



Pisarenko – Enveloppe

- ▶ la majorité des pics tombent dans la bonne bande de fréquences
- ▶ par contre, il faudrait traiter ce spectre de raies, pour espérer pouvoir en tirer quelque information
 - ▶ on pourrait relier (linéairement) les raies : voir la suite
 - ▶ note : on doit faire quelque chose de similaire quand on considère le spectre FFT, sauf que les pics ne sont pas détectés
- ▶ on n'arrive pas facilement à détecter les 20 partiels, même pour des p très grands, à cause des pics parasites qui ont des amplitudes trop grandes
 - ▶ note : en réglant p très finement, sans doute on y parvient

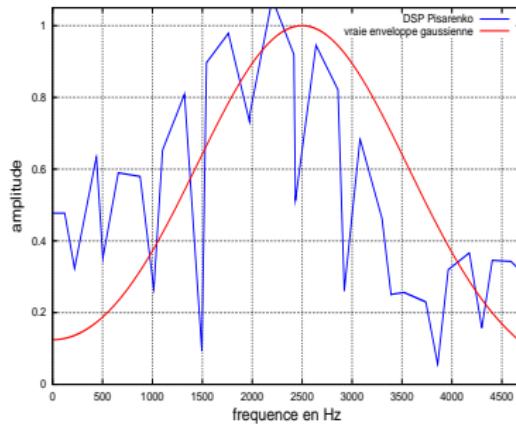
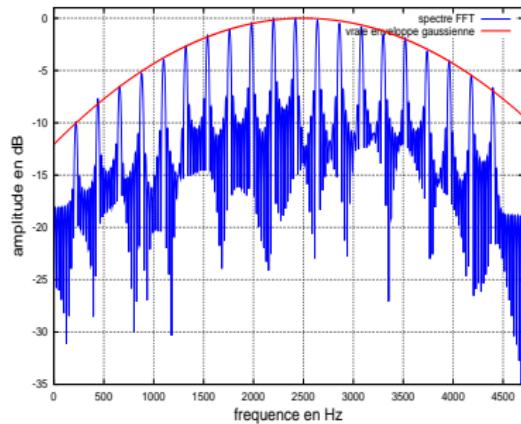
Pisarenko – Enveloppe

► voir :

scriptenveloppisarenko2.m

Pisarenko – Enveloppe

Exemple de spectre FFT et d'enveloppe spectrale Pisarenko ($p = 100$, norme 3) obtenus avec le signal précédent :



Pisarenko – Enveloppe

Note : bien sûr, les pics de la FFT collent bien à la vraie enveloppe, mais par contre il faut encore détecter automatiquement ces pics, pour obtenir l'enveloppe spectrale !

Mesures :

- ▶ on calcule entre 0 et 4700 Hz la différence pour chaque fréquence entre DSP obtenue et vraie enveloppe gaussienne
⇒ la somme normalisée donne la distance entre les 2
- ▶ on fait cette mesure un grand nombre de fois : le résultat dépend des phases respectives des partiels
- ▶ on fait cette mesure pour différents ordres p
- ▶ ON PREND LA MÉDIANE AU LIEU DE LA MOYENNE, À CAUSE DES CAS ABERRANTS

Pisarenko – Enveloppe – Norme 3

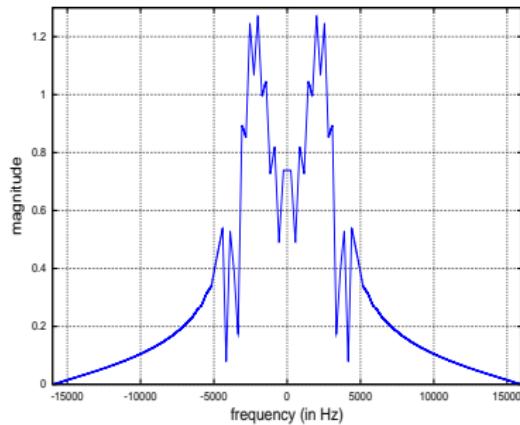
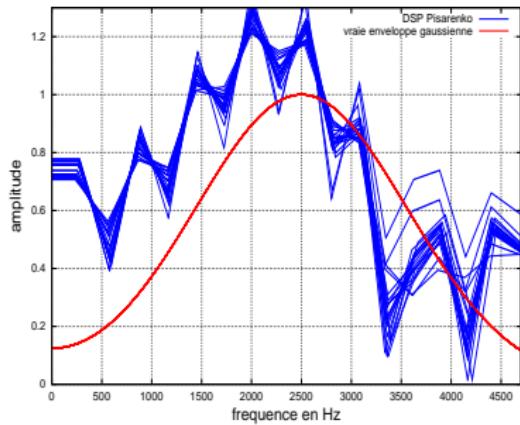
p	N	min	max	median	σ
10	100	1.590579	1.908460	1.781442	0.088149
50	100	0.444954	1.545431	0.488974	0.225396
75	100	0.335142	5.859627	0.470232	0.598085
80	100	0.338674	5.744589	0.422474	0.819707
85	100	0.371058	7.743117	0.538990	1.122852
90	100	0.338030	11.424876	0.501638	1.670231
95	100	0.345127	2.728425	0.423910	0.491146
100	100	0.314998	29.616454	0.392465	4.016689
105	100	0.328754	5.526777	0.410507	0.814055
110	100	0.310638	55.374879	0.407467	5.679361
115	100	0.299414	2.868429	0.410344	0.525649
120	100	0.352688	6.906477	0.418548	1.203819

Pisarenko – Enveloppe

- ▶ la qualité des enveloppes spectrales obtenues est à peu près équivalente à celle des enveloppes spectrales obtenues avec les modèles AR (la normalisation 2 étant quasi impossible à réaliser en pratique)
- ▶ il n'y a pas forcément besoin de normaliser la DSP de Pisarenko, contrairement à ce qui se passe avec les DSP AR
- ▶ sur la figure suivante, on peut le voir
- ▶ ça explique qu'il y ait besoin de prendre la médiane pour limiter les effets des cas aberrants : parfois, l'amplitude de la DSP est trop grande

Pisarenko – Enveloppe

$p = 50$, norme 3 :



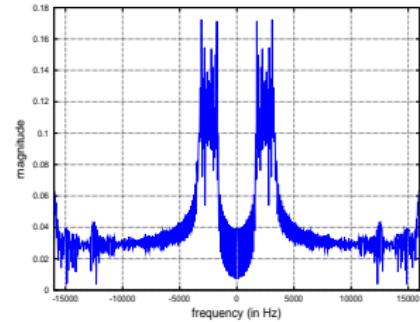
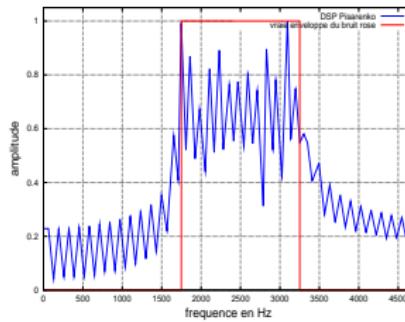
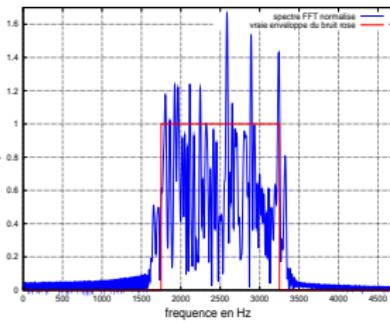
Pisarenko – Bruit rose

Le bruit rose considéré :

- ▶ fréquence centrale du bruit rose : $f_c = 2500 \text{ Hz}$
- ▶ largeur de la bande : 1500 Hz
- ▶ l'amplitude du bruit vaut 1 dans la bande, et 0 donc en-dehors
 - ▶ voir transparent suivant
- ▶ on fait d'abord un bruit blanc, qu'on filtre après coup
- ▶ la longueur du signal est toujours 1280 échantillons, et la fréquence d'échantillonnage 32000 Hz

Pisarenko – Bruit rose

Exemple de spectre FFT et d'enveloppe spectrale de Pisarenko ($p = 250$, norme 1 et norme 3) obtenus avec le signal précédent :



Pisarenko – Bruit rose

Mesures :

- ▶ on calcule entre 0 Hz et 4700 Hz la différence pour chaque fréquence entre la DSP normalisée obtenue et la vraie enveloppe du bruit rose \Rightarrow la somme normalisée donne la distance entre les deux
- ▶ on fait cette mesure un grand nombre de fois : le résultat dépend du tirage initial du bruit blanc
- ▶ on fait cette mesure pour différents ordres p , qu'il faut prendre très grands (du coup, le temps de calcul devient notable)
- ▶ et cette fois on normalise les DSP obtenues en fixant leur maximum à 1 [norme 1]

Pisarenko – Bruit rose

► voir :

scriptpinkpisarenko.m

Pisarenko – Bruit rose – Norme 1

<i>p</i>	<i>N</i>	min	max	median	σ
100	100	0.548114	1.263923	1.140337	0.185918
150	100	0.539377	1.220611	1.110336	0.185350
200	100	0.501536	1.166908	1.088153	0.166405
250	100	0.523339	1.153725	1.070878	0.191676
300	100	0.619697	1.152498	1.076618	0.151215
350	100	0.581091	1.126444	0.992152	0.157461
450	100	0.635147	1.122176	1.005800	0.129507

Résumé des résultats

	Lev.-Durb.	Burg	Marple	Pisarenko	FFT
1 sinus	$p \geq 4$	$p \geq 2$	$3 \leq p \leq 10$ $\sigma^2 = 1e^{-4}$ $4 \leq p \leq 40$ $\sigma^2 = 1e^{-3}$	$p \geq 1$	oui
2 sinus	$p = 250 : 63\%$ $p = 265 : 88\%$ $p = 300 : 100\%$	$p = 4 : 99.68\%$ $p = 5 : 100\%$ $p = 6 : 97\%$ $p = 7 : 7\%$	$p = 90 : 16\%$ $p = 100 : 83\%$ $p = 110 : 93\%$ $p = 120 : 86\%$	méth. autre $p = 40 : 33\%$ $p = 80 : 60\%$ $p = 100 : 77\%$ $p = 150 : 94\%$	Black. 32 % Hann. 40 % (Rect. 64 % méth. autre)
env.	$n1 : e > 0.318$ $n2 : e > 0.305$	pas très efficace	$n1 : e > 0.337$ $n2 : e > 0.310$	$n3 : e > 0.392$	détection pics à faire
20 partiels	oui ($p = 200$)	oui ($p = 100$)	oui ($p = 140$)	dur	oui
bruit rose	$n1 : e > 0.568$ $n2 : e > 0.367$ $n3 : e > 0.378$	$n1 : e > 0.708$ $n2 : e > 0.456$ $n3 : e > 0.493$	$n1 : e > 0.520$ $n2 : e > 0.344$	$n1 : e > 0.99$	$e \simeq 0.5$

Pisarenko – Les amplitudes (norme 3)

- ▶ *mypisarenko.m* : l'amplitude du sinus est 1, et on obtient un pic d'amplitude 1
- ▶ *scriptdoublonpisarenko.m* et *scriptdoublonpisarenko2.m* : l'amplitude de chaque sinus vaut 1, et on obtient des pics d'amplitude du bon ordre de grandeur
- ▶ *scriptenveloppisarenko.m* et *scriptenveloppisarenko2.m* : l'amplitude du plus grand partiel est 1, et on obtient un maximum du même ordre de grandeur
- ▶ *scriptpinkpisarenko.m* : l'amplitude dans la bande du bruit rose est 1, et on obtient une DSP d'amplitude trop petite (de l'ordre de 0.1)

Pisarenko – Le code matlab/octave

- ▶ j'ai mis le code de *mypisarenko.m* sur mon site :
[http://www.metz.supelec.fr/metz/personnel/
rossignol/mypisarenko.m](http://www.metz.supelec.fr/metz/personnel/rossignol/mypisarenko.m)
- ▶ il s'agit encore une fois que vous regardiez et interprétriez par vous-même ce qui se passe, pour différents signaux ; que vous exploriez les limitations et les qualités des diverses méthodes

Pisarenko – Le code en C

- ▶ dans une perspective d'utilisation de ces méthodes d'analyse spectrale en temps réel, il faudrait traduire ce code en C
- ▶ si on examine le code matlab/octave, on voit qu'il y a des complications ; on a besoin :
 - ▶ de l'autocorrélation (difficulté moindre)
 - ▶ de l'inversion de matrice
 - ▶ de l'extraction des racines d'un polynôme
 - ▶ éventuellement de l'extraction des valeurs et vecteurs propres
 - ▶ éventuellement de l'interpolation (cas linéaire : difficulté moindre)
- ▶ voir des bibliothèques pré-existantes pour le calcul matriciel, ou alors fouiner dans les Numerical Recipes
- ▶ bien sûr, il faudrait aussi considérer tous les problèmes liés à l'acquisition des signaux

- ▶ Introduction
- ▶ Modèles AR
- ▶ Pisarenko
- ▶ MUSIC/ESPRIT

MUSIC

- ▶ MUSIC (MULTiple SIgnal Characterization) est une extension directe de Pisarenko
- ▶ en ce qui concerne MUSIC, au lieu de ne considérer que la plus petite valeur propre de la matrice d'autocorrélation de taille M , on considère les $M - p$ plus petites
- ▶ on décompose ensuite cette matrice d'autocorrélation en deux sous-espaces, qui sont orthogonaux (ça se prouve, en algèbre linéaire) :
 - ▶ un sous-espace signal de dimension p , composé des vecteurs propres \mathbf{w}_k correspondant aux plus grandes valeurs propres
 - ▶ et un sous-espace-bruit de dimension $M - p$, composé des vecteurs propres \mathbf{v}_k correspondant aux plus petites valeurs propres (sensées être égales à σ^2)

MUSIC

- ▶ et au lieu de ne projeter que sur le vecteur propre correspondant à la plus petite valeur propre, on projette sur le sous-espace bruit (ce qu'on aurait pu faire au lieu de chercher les racines de \mathbf{a} pour Pisarenko)
- ▶ comme pour Pisarenko, donc, on a : $\hat{R}_{xx} = R_{ss} + R_{nn}$, ou encore $R_{ss} = \hat{R}_{xx} - \sigma^2 \mathbf{Id}$
- ▶ pour Pisarenko, les matrices étaient de dimension $(2m + 1) \times (2m + 1)$ (m étant le nombre de raies réelles considérées)
- ▶ pour MUSIC, elles sont de dimension $M \times M$, avec M a priori beaucoup plus grand que $2m + 1$

MUSIC

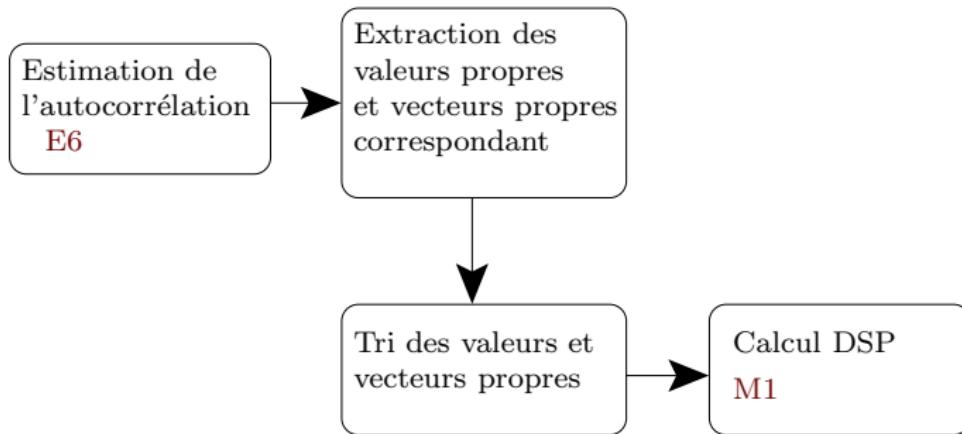
- ▶ la DSP est alors égale à :

$$P(f) = \frac{1}{\mathbf{e}^H(f) \left[\sum_{k=p+1}^M \mathbf{v}_k \mathbf{v}_k^H \right] \mathbf{e}(f)} \quad \text{M1}$$

avec : $\mathbf{e}(f) = \left(1, \cos\left(2\pi \frac{f}{f_e}\right), \dots, \cos\left(2\pi \frac{(M-1)f}{f_e}\right) \right)$ (si les raies étaient complexes, on aurait des exponentielles)

- ▶ note : de la même façon qu'on peut utiliser ceci pour obtenir une DSP avec Pisarenko au lieu d'un spectre de raies, on pourrait pour MUSIC extraire des raies au lieu d'une vraie DSP ; ce serait la méthode appelée root-MUSIC

MUSIC – Résumé



MUSIC – Code matlab

```
function [ff, mydsp] = mymusic(xx, pp, MM, fe)
if (MM<=pp)      fprintf(1, 'Il faut absolument MM>pp !!!');      break;      end ;
MM1=MM-1;xx = xx-mean(xx);
acf = xcorr(xx, MM1, 'unbiased'); %%% corrélation
IMM=length(acf);
rrr = acf(MM1+1 :IMM)';
for ii=1 :MM1
    rrr = [rrr acf(MM1+1-ii :IMM-ii)'];
end ;
rrr = rrr';
[v, lambda] = eig(rrr); %%% méthode directe pour trouver toutes les valeurs propres
lamb = diag(lambda);
[vl,pl] = sort(lamb,'descend');
df=0.9765625; %%% densité spectrale de puissance – la dsp est calculée tous les df Hz
ff=fe/2 :df :fe/2;
mydsp=zeros(length(ff),1);
deni=zeros(MM,MM);
for ii=pp+1 :MM      deni = deni + v( :,pl(ii))*conj(v( :,pl(ii)))';      end ;
for ii=1 :length(ff)
    ee = cos(2*pi*ff(ii)*[0 :MM1]/fe);
    den = conj(ee)*deni*ee';
    mydsp(ii) = abs(1/den);
end ;
mydsp = mydsp-min(mydsp); %%% on enlève éventuellement une composante non nulle
```

MUSIC – Premiers tests

On va faire quelques tests, avec des signaux (sons) simples pour vérifier qu'on obtient bien des densités spectrales de puissance intéressantes. Les 4 tests qu'on considère ici sont :

- ▶ on a une sinusoïde pure \Rightarrow est-ce qu'on arrive à la mettre en évidence ?
- ▶ on a deux sinusoïdes proches en fréquence \Rightarrow est-ce qu'on arrive à les discriminer ?
- ▶ on a un signal compliqué : une somme de sinusoïdes harmoniques \Rightarrow est-ce qu'on arrive à obtenir une enveloppe spectrale, modélisant ce signal, et/ou les n partiels ?
- ▶ on a un signal compliqué : un bruit rose \Rightarrow est-ce qu'on arrive à obtenir une enveloppe spectrale, modélisant ce signal ?

MUSIC – Une sinusoïde

mymusic_matlab.m

Paramètres de l'exemple :

- ▶ $f_e = 32000$ Hz
- ▶ signal : une sinusoïde présente
 - ▶ de fréquence 440 Hz
 - ▶ et de phase aléatoire, tirée entre 0 et 2π
 - ▶ de taille 1280 échantillons (40 ms)
- ▶ $p = 2$ et $M = 10$
 - ▶ ATTENTION : ON A À PRÉSENT CES 2 PARAMÈTRES À FAIRE VARIER

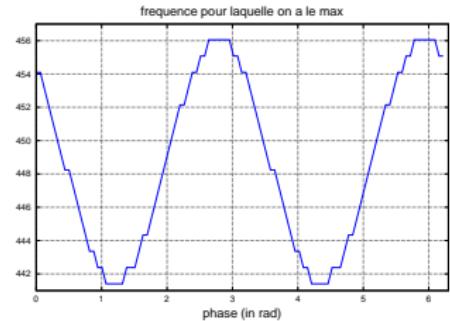
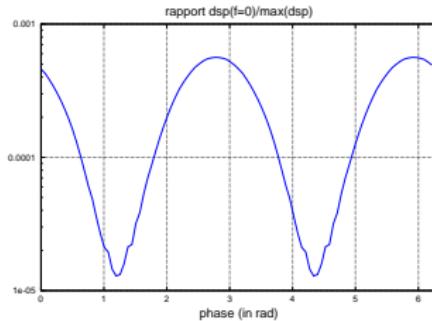
MUSIC – Une sinusoïde

scriptphasemusic.m

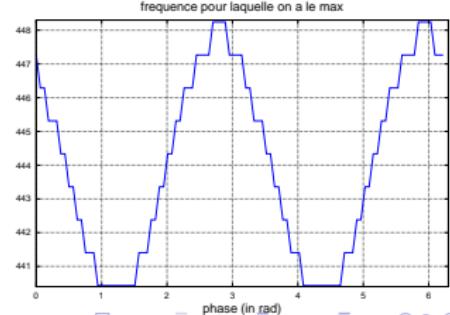
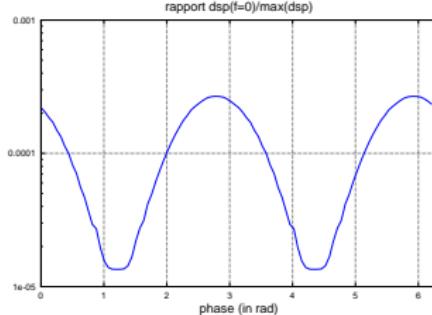
- ▶ on regarde ce qui se passe en fonction de la phase de la sinusoïde
- ▶ on mesure le rapport entre l'amplitude du spectre de raies en $f = 0$ et son amplitude maximum (sensée se trouver à $\pm 440 \text{ Hz}$)
- ▶ on regarde où se trouve en réalité l'amplitude maximum du spectre de raies
- ▶ ce pour différents ordres : $p = 1, p = 2, p = 3, \dots$ et pour différents $M : M = 2, M = 3, M = 4, \dots$ (en notant qu'il faut absolument $M > p$)

MUSIC – Une sinusoïde

$p = 2, M = 5 :$



$p = 2, M = 10 :$



MUSIC –

- ▶ on donne ci-dessous le nombre de fois sur 100 où la fréquence du maximum a été trouvée entre 400 Hz et 480 Hz

M	3	4	5	6	7	10	105	110	150
$p = 1$	0	0	0	0	0	0	0	100	100
$p = 2$	0	100	100	100	100	100	etc	etc	etc
$p = 3$	—	0	100	100	100	100	etc	etc	etc
$p = 4$	—	—	0	100	100	100	etc	etc	etc
$p = 5$	—	—	—	0	100	100	etc	etc	etc
$p = 6$	—	—	—	—	0	100	100	etc	etc

- ▶ en théorie, il faut plutôt que p soit pair (signal réel) ; en pratique, ça ne semble pas aussi important
- ▶ une valeur du paramètre p trop grande ne semble pas problématique
- ▶ par contre, ne pas hésiter à prendre M bien plus grand que p

MUSIC – Intérêt

- ▶ est-ce que l'algorithme de MUSIC présente le même intérêt que les autres méthodes en termes de haute-résolution ?
- ▶ on refait les mêmes tests qu'avec les méthodes AR, Pisarenko et la FFT
 - ▶ 2 sinusoïdes proches, séparées de 26 Hz (440 Hz et 466 Hz) et de même amplitude, etc. : voir transparent suivant
- ▶ et on fait varier les ordres p et M
 - ▶ on part d'ordres aussi petits que possible

MUSIC – Intérêt

- ▶ la longueur du signal est 1280 échantillons (une seule trame de 40 ms)
- ▶ sinusoïdes de fréquences 440 Hz et 466 Hz
- ▶ la DSP est calculée sur 32768 points
- ▶ méthode de détection simple : on détecte les maximums et minimums locaux entre $f_{min} - 40 \text{ Hz}$ et $f_{max} + 40 \text{ Hz}$
- ▶ puis on vérifie qu'il y a bien 2 maximums locaux, 1 minimum local, et que celui-ci se situe entre les 2 maximums locaux
- ▶ le résultat obtenu dépend de la phase respective des 2 sinus
- ▶ il dépend aussi des ordres p et M du modèle MUSIC

MUSIC – Intérêt

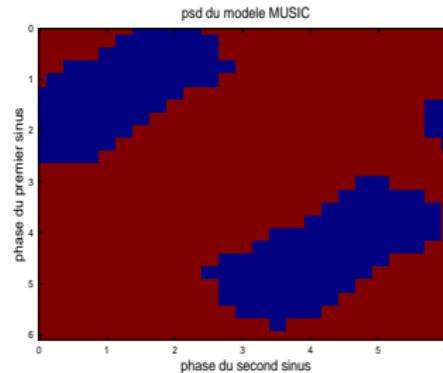
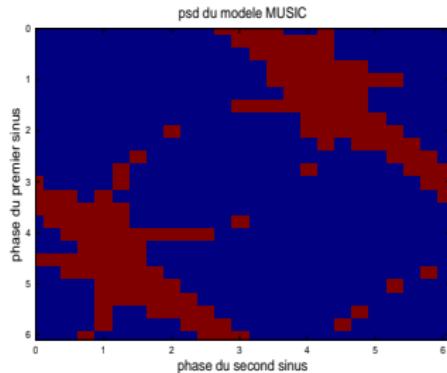
► voir :

scriptdoublonmusic.m

MUSIC – Intérêt

MUSIC ($p = 4$, $M = 90$) : 23.04 % de succès

MUSIC ($p = 4$, $M = 100$) : 73.44 % de succès



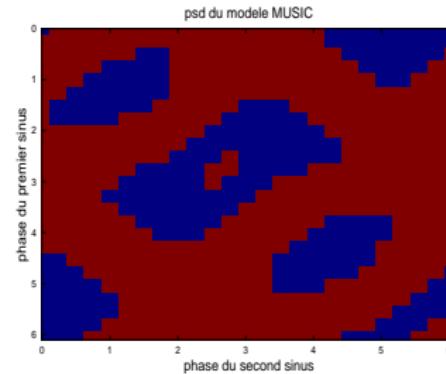
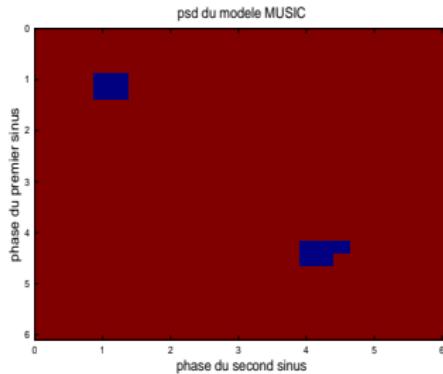
En rouge : les 2 sinusoïdes ont été séparées

En bleu : les 2 sinusoïdes n'ont pas été séparées

MUSIC – Intérêt

MUSIC ($p = 4$, $M = 110$) : 98.56 % de succès

MUSIC ($p = 4$, $M = 120$) : 68.00 % de succès



En rouge : les 2 sinusoïdes ont été séparées

En bleu : les 2 sinusoïdes n'ont pas été séparées

MUSIC – Intérêt

- ▶ ça marche pour $p = 4$, qui est théoriquement l'ordre minimum qui permet de détecter 2 sinusoïdes (notre cas ici)
- ▶ on atteint quasi 100 % de bonnes détections
- ▶ on fait mieux qu'avec la FFT, pour laquelle, en tirant sur la corde, on obtient au mieux 64 % de bonnes détections
- ▶ mise en place de méthodes d'estimation des ordres p et M à utiliser
- ▶ il faudrait regarder ce qui se passe pour d'autres ordres p que 4 ; mais l'étude se complique du fait qu'il faut aussi s'occuper de M

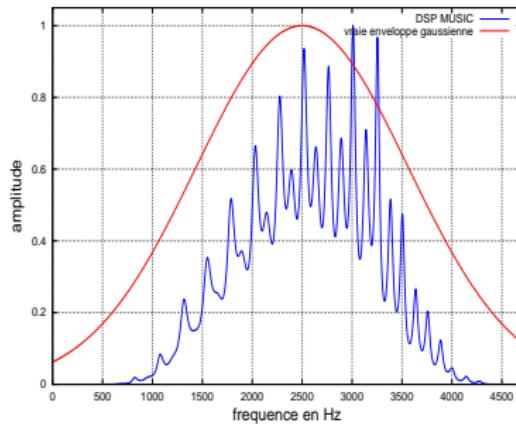
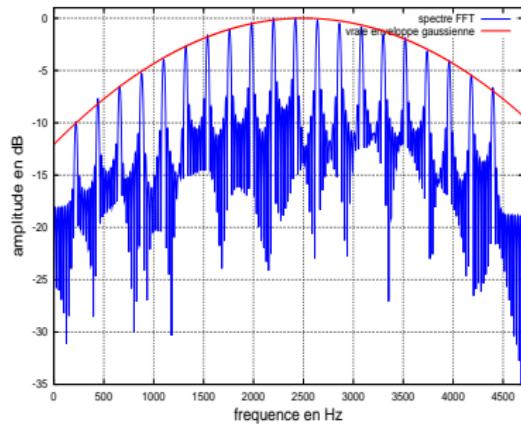
MUSIC – Enveloppe

On a un signal harmonique (composé d'une somme de partiels harmoniques). Le signal considéré :

- ▶ fréquence fondamentale : $f_0 = 220 \text{ Hz}$
- ▶ nombre de partiels : 20
- ▶ amplitudes des partiels : elles suivent une gaussienne centrée en $f_{max} = 2500 \text{ Hz}$ et d'écart-type 1500
 - ▶ voir transparent suivant
- ▶ phases des partiels : tirées aléatoirement entre 0 et 2π
- ▶ la longueur du signal est toujours 1280 échantillons, et la fréquence d'échantillonnage 32000 Hz

MUSIC – Enveloppe

Exemple de spectre FFT et d'enveloppe spectrale MUSIC ($p = 30$, $M = 124$, norme 1) obtenus avec le signal précédent :



MUSIC – Enveloppe

Note : bien sûr, les pics de la FFT collent bien à la vraie enveloppe, mais par contre il faut encore détecter automatiquement ces pics, pour obtenir l'enveloppe spectrale !

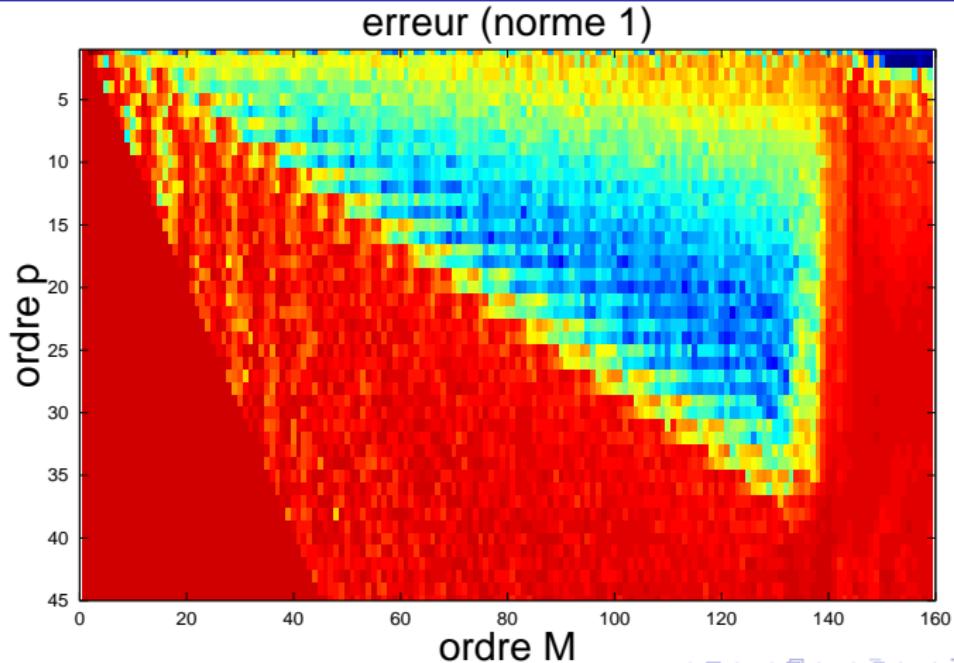
Mesures :

- ▶ on calcule entre 0 Hz et 4700 Hz la différence pour chaque fréquence entre la DSP normalisée obtenue et la vraie enveloppe gaussienne \Rightarrow la somme normalisée donne la distance entre les deux
- ▶ on devrait faire cette mesure un grand nombre de fois : le résultat dépend des phases respectives des partiels ; mais ce n'est pas ais \acute{e} car il faut << optimiser >> p et M
- ▶ on fait cette mesure pour différents ordres p et différents ordres M

MUSIC – Enveloppe

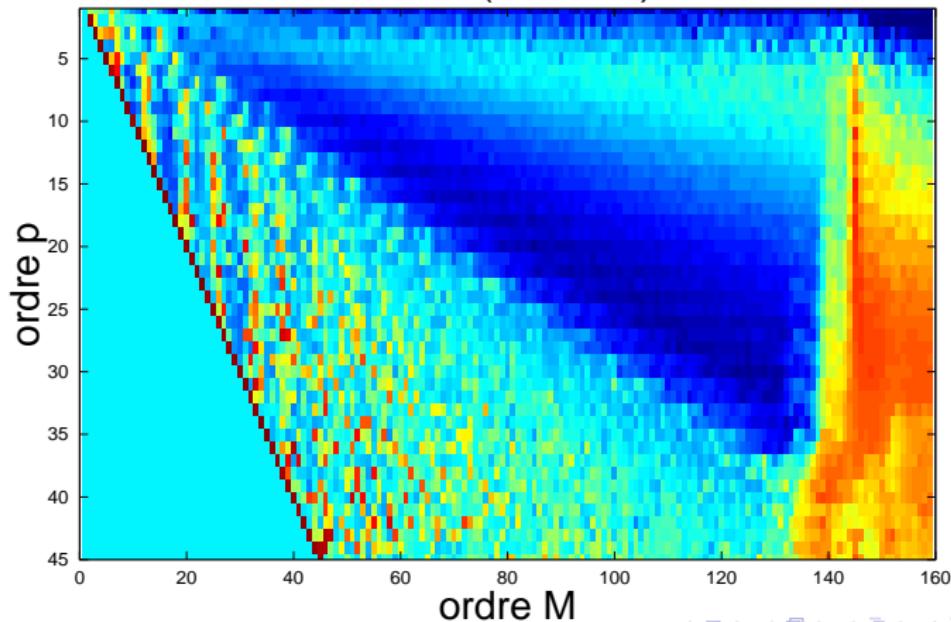
- ▶ voir : *scriptenvelopmusic.m*; avec ce script, on essaie de déterminer une plage de p et de M qui semblent promettre de bonnes performances
 - ▶ voir les 3 figures suivantes
 - ▶ pour la norme 1, on obtient les plus basses erreurs sur une ligne qui va de 14 à 25 pour p et de 70 à 130 pour M , soit environ $M = \text{round}(5.45p + 6.36)$
 - ▶ pour la norme 2, on obtient les plus basses erreurs sur une ligne qui va de 15 à 32 pour p et de 70 à 130 pour M , soit environ $M = \text{round}(3.53p + 17.06)$
- ▶ voir : *scriptenvelopmusic.m*; avec ce script, on vérifie les performances obtenues pour les p et M déterminés ci-dessus

MUSIC – Enveloppe

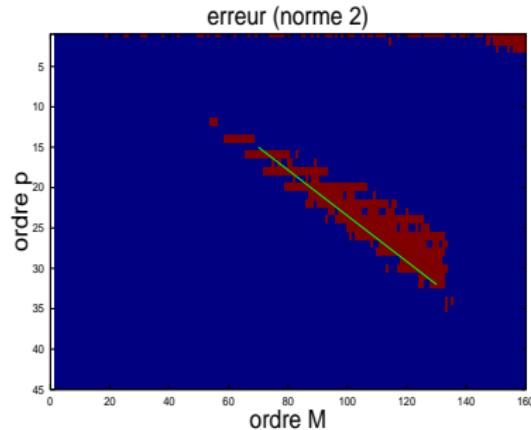
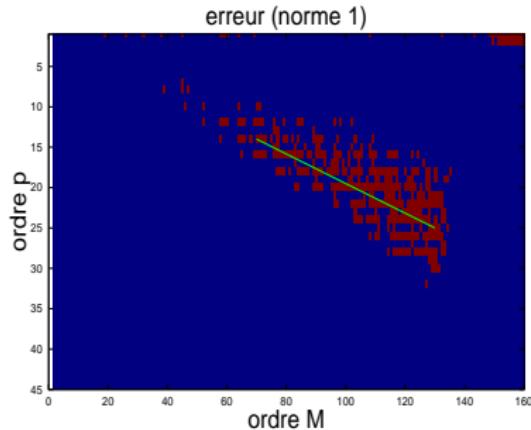


MUSIC – Enveloppe

erreur (norme 2)



MUSIC – Enveloppe



Norme 1, en rouge les points pour lesquels $e < 0.6$, et en vert ligne considérée ; Norme 2, en rouge les points pour lesquels $e < 0.55$, et en vert ligne considérée

MUSIC – Enveloppe – Norme 1

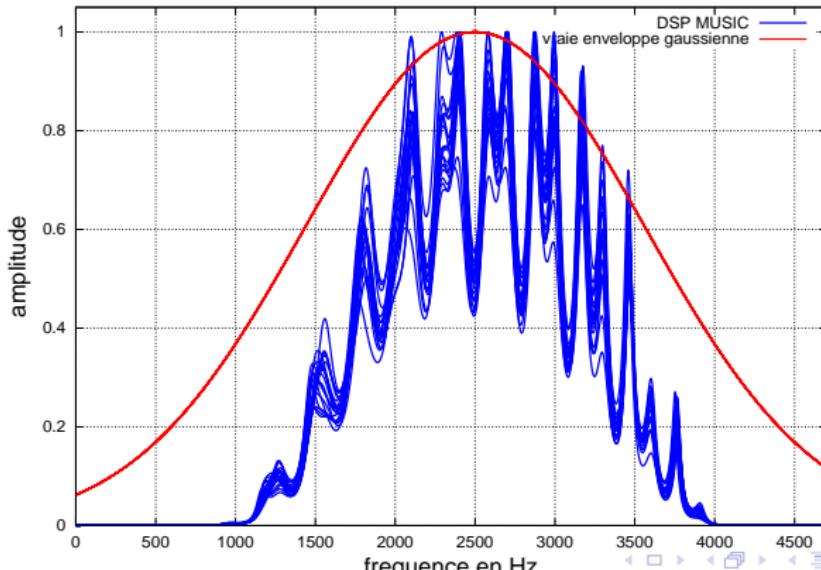
<i>p</i>	<i>M</i>	<i>N</i>	min	max	mean	σ
14	70	100	0.509850	0.622475	0.564676	0.026462
18	92	100	0.500620	0.630420	0.577501	0.027923
19	103	100	0.522840	0.633196	0.585535	0.023074
20	100	100	0.536501	0.637216	0.581840	0.021228
20	103	100	0.467587	0.589208	0.526620	0.025636
20	106	100	0.522780	0.640786	0.580417	0.022612
21	103	100	0.538719	0.684286	0.601384	0.027972
24	125	100	0.499441	0.621006	0.578100	0.023651

MUSIC – Enveloppe – Norme 2

p	M	N	min	max	mean	σ
16	73	100	0.484051	0.524293	0.506100	0.008708
20	88	100	0.470903	0.530377	0.496924	0.010662
25	105	100	0.490105	0.542541	0.522350	0.010897
29	120	100	0.467261	0.560925	0.490047	0.017671
30	120	100	0.442954	0.558599	0.524654	0.019378
30	123	100	0.443983	0.489868	0.461553	0.009933
30	126	100	0.435328	0.491180	0.466774	0.009288
35	141	100	1.391159	1.582867	1.549739	0.039321
40	158	100	1.584514	1.611521	1.597714	0.005981

MUSIC – Enveloppe – $p = 20$, $M = 103$, norme 1

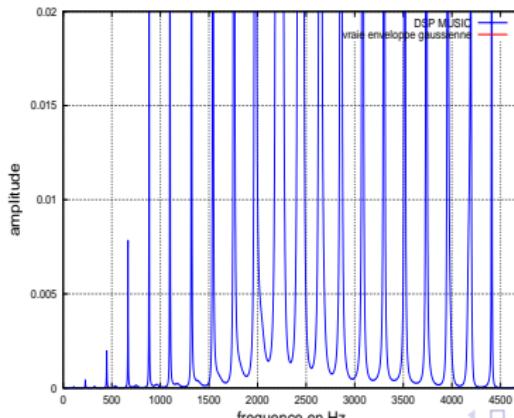
L'amplitude des DSP MUSIC semble sous-estimée



MUSIC – Enveloppe – Norme 1

Est-ce qu'en augmentant suffisamment les ordres p et M on arrive à récupérer les 20 partiels ?

Oui ! Ainsi, pour $p = 50$ et $M = 150$, on obtient par exemple, en zoomant sur y :



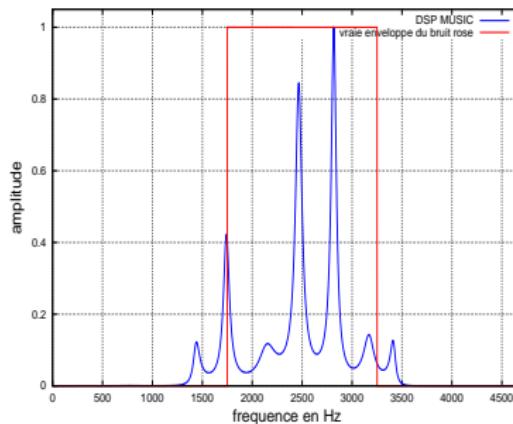
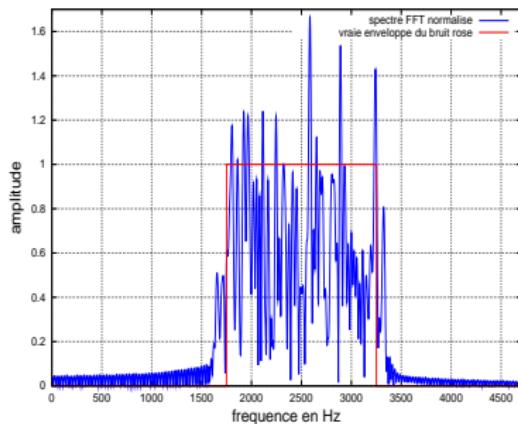
MUSIC – Bruit rose

Le bruit rose considéré :

- ▶ fréquence centrale du bruit rose : $f_c = 2500 \text{ Hz}$
- ▶ largeur de la bande : 1500 Hz
- ▶ l'amplitude du bruit vaut 1 dans la bande, et 0 donc en-dehors
 - ▶ voir transparent suivant
- ▶ on fait d'abord un bruit blanc, qu'on filtre après coup
- ▶ la longueur du signal est toujours 1280 échantillons, et la fréquence d'échantillonnage 32000 Hz

MUSIC – Bruit rose

Exemple de spectre FFT et de DSP MUSIC ($p = 12$, $M = 34$, norme 1) obtenus avec le signal précédent :



MUSIC – Bruit rose

Note : la normalisation du spectre FFT est parfaitement contrôlée
(considérations énergétiques concernant les spectres continus)

Mesures :

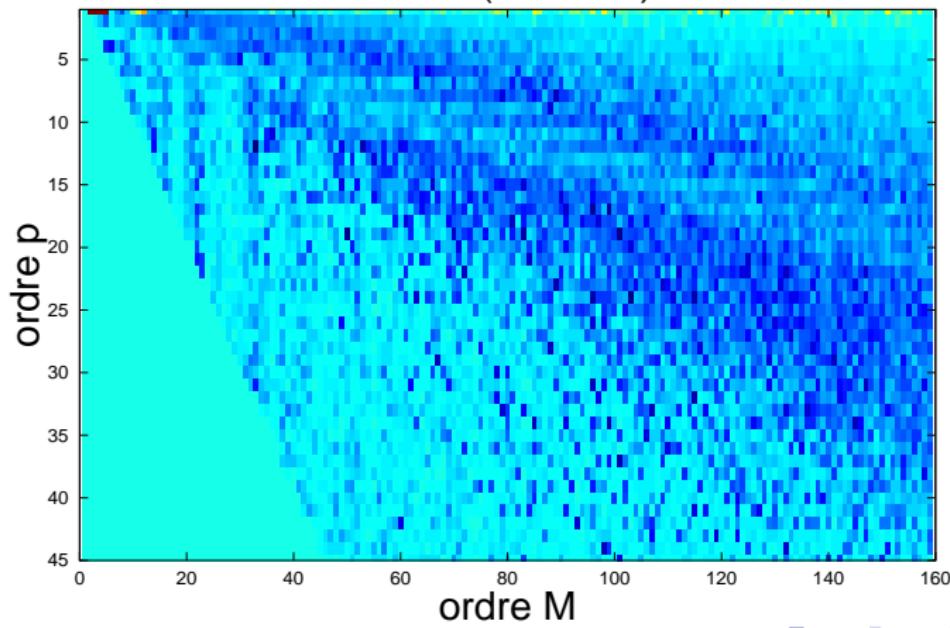
- ▶ on calcule entre 0 Hz et 4700 Hz la différence pour chaque fréquence entre la DSP normalisée obtenue et la vraie enveloppe du bruit rose \Rightarrow la somme normalisée donne la distance entre les deux
- ▶ on fait cette mesure un grand nombre de fois : le résultat dépend du tirage initial du bruit blanc
- ▶ on fait cette mesure pour différents ordres p

MUSIC – Bruit rose

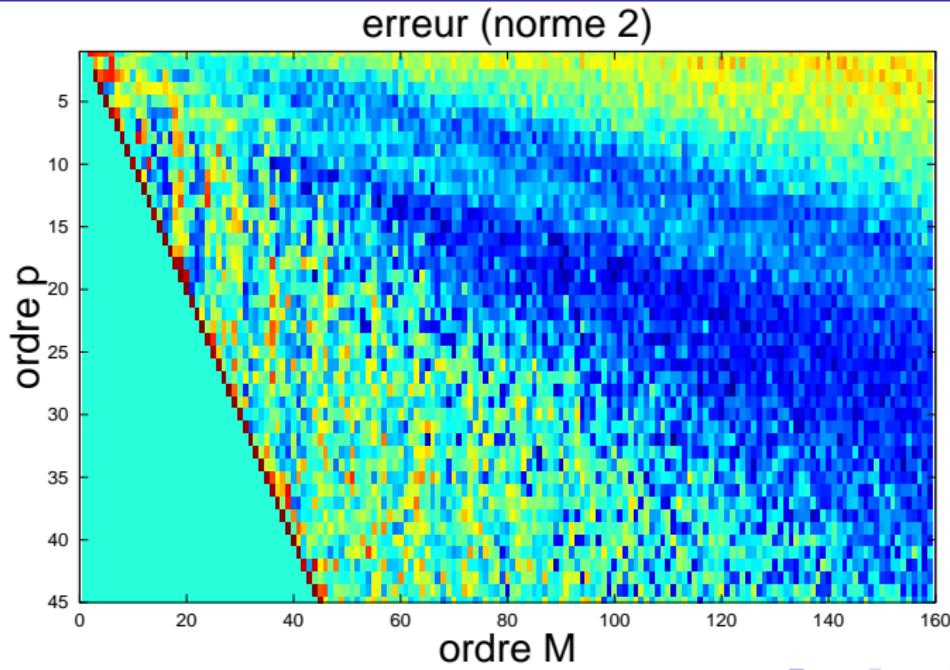
- ▶ voir : *scriptpinkmusic.m*; avec ce script, on essaie de déterminer une plage de p et de M qui semblent promettre de bonnes performances
 - ▶ voir les 3 figures suivantes
 - ▶ pour la norme 1, on obtient les plus basses erreurs dans une grande zone, disons aux alentours de $p = 20$ et pour $M > 80$
 - ▶ pour la norme 2, on obtient les plus basses erreurs dans la même zone que pour la norme 1
- ▶ voir : *scriptpinkmusic.m*; avec ce script, on vérifie les performances obtenues pour les p et M déterminés ci-dessus

MUSIC – Bruit rose

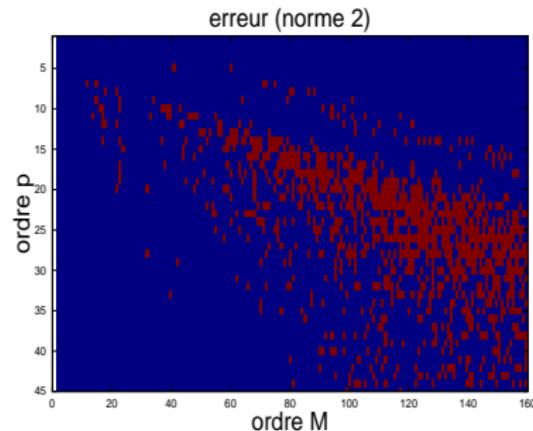
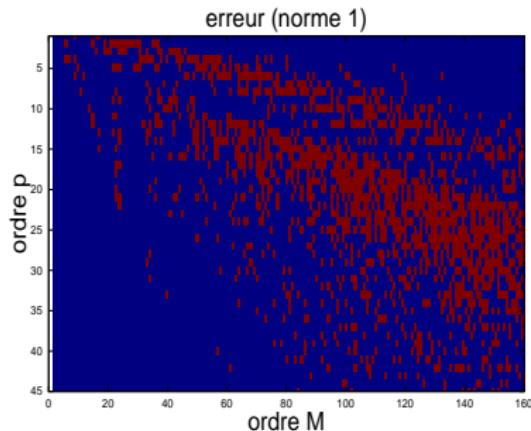
erreur (norme 1)



MUSIC – Bruit rose



MUSIC – Bruit rose



Norme 1, en rouge les points pour lesquels $e < 0.8$; Norme 2, en rouge les points pour lesquels $e < 0.6$

MUSIC – Bruit rose – Norme 1

p	M	N	min	max	mean	σ
12	34	100	0.560621	0.995284	0.852408	0.092411
20	80	100	0.481215	0.946958	0.783618	0.102518
22	100	100	0.482145	0.942702	0.782671	0.087376
24	120	100	0.559727	0.917416	0.746229	0.082983
26	120	100	0.461708	0.939691	0.778136	0.091491
26	125	100	0.512055	0.925746	0.755913	0.089015

⇒ recherche plus exhaustive à faire

MUSIC – Bruit rose – Norme 2

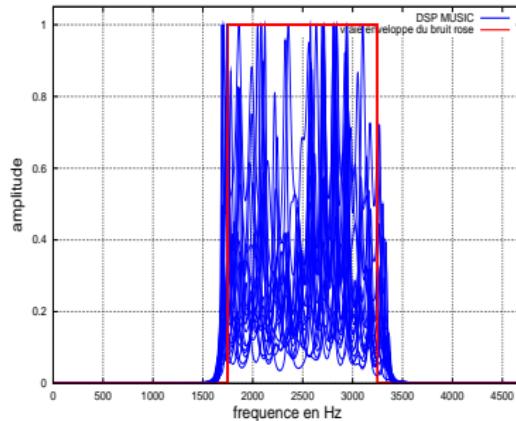
p	M	N	min	max	mean	σ
13	57	100	0.298643	0.978390	0.618053	0.156113
20	80	100	0.386085	1.024465	0.615883	0.127276
22	100	100	0.376287	0.888439	0.575620	0.102686
24	120	100	0.357096	0.808647	0.527289	0.074928
26	120	100	0.324876	0.842905	0.553917	0.110521
26	125	100	0.408436	0.906958	0.568695	0.092150

⇒ recherche plus exhaustive à faire

MUSIC – Bruit rose – Norme 1

L'amplitude des DSP MUSIC semble sous-estimée

$p = 24, M = 120 :$



Résumé des résultats

	Lev.-Durb.	Burg	Marple	Pisarenko	MUSIC	FFT
1 sin	$p \geq 4$	$p \geq 2$	$3 \leq p \leq 10$ $\sigma^2 = 1e^{-4}$ $4 \leq p \leq 40$ $\sigma^2 = 1e^{-3}$	$p \geq 1$	$p = 1 \Rightarrow M \geq 110$ $p \geq 2 \Rightarrow M \geq p + 2$	oui
2 sin	$p=250$ 63% $p=265$ 88% $p=300$ 100%	$p=4$ 99.68% $p=5$ 100% $p=6$ 97% $p=7$ 7%	$p=90$ 16% $p=100$ 83% $p=110$ 93% $p=120$ 86%	mét. autre $p=40$ 33% $p=80$ 60%	$p=4 M=90$ 23% $p=4 M=100$ 73% $p=4 M=110$ 98.6% $p=4 M=120$ 68%	Black. 32% Hann. 40% Rect. 64% mét. autre
env	$e_1 > 0.318$ $\bar{e}_2 > 0.305$	pas très efficace	$e_1 > 0.337$ $\bar{e}_2 > 0.310$	$e_3 > 0.392$	$e_1 > 0.527$ $\bar{e}_2 > 0.462$	détection pics à faire
20 sin	oui ($p=200$)	oui ($p=100$)	oui ($p=140$)	dur	oui ($p=50 M=150$)	oui
rose	$\bar{e}_1 > 0.568$ $\bar{e}_2 > 0.367$ $e_3 > 0.378$	$\bar{e}_1 > 0.708$ $\bar{e}_2 > 0.456$ $e_3 > 0.493$	$\bar{e}_1 > 0.520$ $\bar{e}_2 > 0.344$	$\bar{e}_1 > 0.99$	$\bar{e}_1 > 0.746$ $\bar{e}_2 > 0.527$	$e \simeq 0.5$

MUSIC – Les amplitudes (norme 3)

- ▶ *mymusic_matlab.m* : l'amplitude du sinus est 1, et on obtient un pic gigantesque d'amplitude de l'ordre de 10^6
- ▶ *scriptdoublonmusic.m* : l'amplitude de chaque sinus vaut 1, et on obtient des pics de grandes amplitudes (ordre de grandeur : 250 et 75)
- ▶ *scriptenvelopmusic.m* : l'amplitude du plus grand partiel est 1, et on obtient un maximum dont l'amplitude est de l'ordre de 60
- ▶ *scriptpinkmusic.m* : l'amplitude dans la bande du bruit rose est 1, et on obtient une DSP d'amplitude gigantesque (de l'ordre de 4000)

MUSIC – Le code matlab/octave

- ▶ j'ai mis le code de *mymusic_matlab.m* sur mon site :
[http://www.metz.supelec.fr/metz/personnel/
rossignol/mymusic_matlab.m](http://www.metz.supelec.fr/metz/personnel/rossignol/mymusic_matlab.m)
- ▶ bien sûr, les scripts que j'ai faits sont réservés à mon usage :
je vous ai donné *scriptphaselevinson.m* pour vous inciter à
jouer de votre côté avec ces méthodes d'analyse spectrale

MUSIC – Le code en C

- ▶ dans une perspective d'utilisation de ces méthodes d'analyse spectrale en temps réel, il faudrait traduire ce code en C
- ▶ si on examine le code matlab/octave, on voit qu'il y a des complications ; on a besoin :
 - ▶ de l'autocorrélation (difficulté moindre)
 - ▶ des opérations de manipulation des matrices (produit...)
 - ▶ de l'extraction des valeurs et vecteurs propres
- ▶ voir des bibliothèques pré-existantes pour le calcul matriciel, ou alors fouiner dans les Numerical Recipes
- ▶ bien sûr, il faudrait aussi considérer tous les problèmes liés à l'acquisition des signaux

- ▶ Introduction
- ▶ Modèles AR
- ▶ Pisarenko
- ▶ MUSIC/ESPRIT
- ▶ Prony

Prony

- ▶ on considère souvent que la méthode de Prony est une autre extension possible de Pisarenko
- ▶ ce même si elle a été inventée avant celle de Pisarenko : en effet, la méthode de Prony date de 1795
- ▶ on ne va examiner que la méthode de Prony historique : la méthode étendue (appelée aussi Prony moindres carrés) retombe en grande partie sur la modélisation AR

Prony

- ▶ par contre, la méthode historique consiste à estimer p raies complexes à partir de $2p$ échantillons de signal seulement
- ▶ c'est-à-dire que la taille du signal et l'ordre du modèle sont liés
- ▶ cependant, on va voir que quand même on continuera à considérer des signaux de 1280 échantillons, comme précédemment
- ▶ bien sûr, si le signal est réel, à partir de nos $2p$ échantillons, on récupère $p/2$ raies dans les fréquences positives et leurs $p/2$ contreparties dans les fréquences négatives

Prony

- ▶ le modèle de signal est encore différent ; on utilise :

$$x_n = \sum_{i=1}^p h_i z_i^n \text{ pour } n \geq 0$$

où les h_i sont des amplitudes complexes : $h_i = \rho_i \exp(j\Theta_i)$; et où les z_i sont des exponentielles complexes modélisant des raies amorties : $z_i = \exp(\alpha_i + j2\pi f_i/f_e)$

- ▶ on montre que le signal suit une relation de régression du type de celle du modèle AR (sauf que l'entrée n'est pas du bruit)
- ▶ **point 1.** en effet, on définit le polynôme $P(z) = \prod_{i=1}^p (z - z_i)$,

où les z_i sont les exponentielles vues ci-dessus

Prony

- ▶ alors on peut le développer pour obtenir : $P(z) = \sum_{i=0}^p a_i z^{p-i}$

$$▶ \text{or, on a : } \sum_{i=0}^p a_i x_{k-i} = \sum_{i=0}^p a_i \left(\sum_{j=1}^p h_j z_j^{k-i} \right)$$

- ▶ on peut intervertir les deux sommes et on obtient :

$$\sum_{j=1}^p h_j \left(\sum_{i=0}^p a_i z_j^{k-i} \right) = \sum_{j=1}^p h_j z_j^{k-p} \left(\sum_{i=0}^p a_i z_j^{p-i} \right)$$

- ▶ le terme entre parenthèse est nul par définition du polynôme $P(z)$

Prony

- de ce fait, on a : $\sum_{i=0}^p a_i x_{k-i} = 0$, soit encore :

$$a_0 x_k = - \sum_{i=1}^p a_i x_{k-i}$$

- a_0 est égal à 1 : il suffit d'inspecter $P(z)$ pour s'en convaincre
- alors, à partir de $2p$ échantillons (sur les N qu'on a à notre disposition, n'oublions pas), on peut écrire :

$$\begin{bmatrix} x_{p-1} & x_{p-2} & \dots & x_0 \\ x_p & x_{p-1} & \dots & x_1 \\ \vdots & \vdots & \dots & \vdots \\ x_{2p-2} & x_{2p-3} & \dots & x_{p-1} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix} = - \begin{bmatrix} x_p \\ x_{p+1} \\ \vdots \\ x_{2p-1} \end{bmatrix} \quad \text{Y1}$$

Prony

- ▶ on inverse cette matrice, d'une manière ou d'une autre, pour obtenir le vecteur \mathbf{a}
- ▶ on peut alors calculer les racines du polynôme $P(z)$, pour estimer les \hat{z}_i ;
- ▶ ensuite, on obtient les estimées des coefficients d'amortissement et des fréquences des modes ainsi :

$$\hat{\alpha}_i = \log |z_i| \text{ et } \hat{f}_i = \frac{1}{2\pi} \arctan \left(\frac{\text{Im}(z_i)}{\text{Re}(z_i)} \right)$$

- ▶ **point 2.** reste à estimer les amplitudes complexes
- ▶ pour ce faire, on repart de la définition du modèle

Prony

- en effet, on peut alors former un autre système linéaire :

$$\begin{bmatrix} 1 & 1 & \dots & 1 \\ z_1 & z_2 & \dots & z_p \\ \vdots & \vdots & \dots & \vdots \\ z_1^{p-1} & z_2^{p-1} & \dots & z_p^{p-1} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_p \end{bmatrix} = - \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{p-1} \end{bmatrix} \quad Y2$$

- on inverse cette matrice, d'une manière ou d'une autre, pour obtenir le vecteur \mathbf{h}
- et on obtient les estimées des amplitudes et des phases ainsi :

$$\hat{\rho}_i = |h_i| \text{ et } \hat{\Theta}_i = \arctan \left(\frac{\text{Im}(h_i)}{\text{Re}(h_i)} \right)$$

Prony

- ▶ **point 3.** il reste encore à calculer le spectre de Prony
- ▶ on peut se contenter du spectre de raies, puisqu'on récupère des fréquences et des amplitudes (on obtient le même type de spectres qu'avec la méthode de Pisarenko)
- ▶ ou on peut calculer une DSP ; pour améliorer la résolution, on symétrise le signal x_n de cette façon :

$$x_n = \sum_{i=1}^p h_i z_i^n \text{ pour } n \geq 0$$

$$x_n = \sum_{i=1}^p h_i (z_i^*)^n \text{ pour } n < 0$$

Prony

- ▶ la transformée en Z de ce signal est :

$$X(z) = \sum_{i=1}^p h_i \left(\frac{1}{1 - z_i z^{-1}} - \frac{1}{1 - (z_i^* z)^{-1}} \right)$$

- ▶ et, comme habituellement, la DSP qui en résulte est :

$$S(f) = \left| \frac{1}{f_e} X(\exp(j2\pi f/f_e)) \right|^2 \text{ Y3}$$

- ▶ **point 4.** et finalement il reste à résoudre les cas de mauvais conditionnements pour les deux matrices à inverser

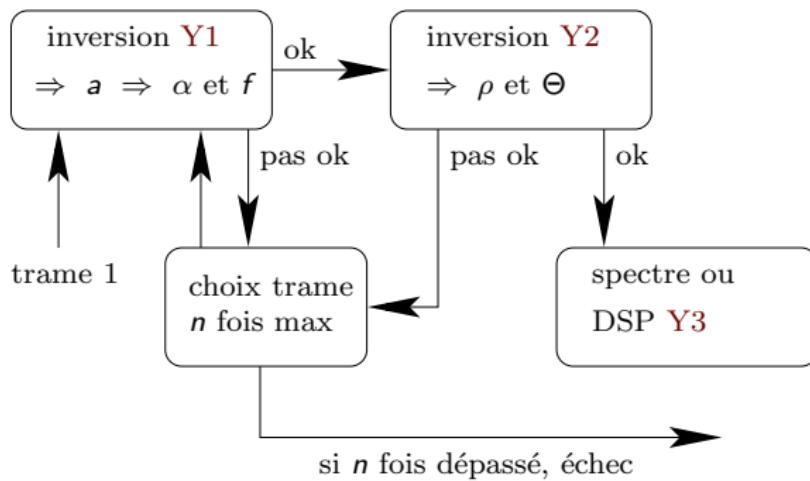
Prony

- ▶ en fait, les matrices à inverser (voir \mathbf{Y}_1 et \mathbf{Y}_2) ne sont pas forcément bien conditionnées, et donc ne sont pas facilement inversibles
- ▶ du coup, quand ça arrive, on va prendre une autre partie du signal que les $2p$ premiers échantillons sur $N = 1280$; et travailler sur cette nouvelle tranche
- ▶ c'est ce qui a été implanté dans *myprony-matlab.m*
- ▶ en cas de mauvais conditionnement d'une des 2 matrices, on choisit de manière aléatoire une autre tranche de signal, parmi les 1280 échantillons disponibles

Prony

- ▶ cette opération est répétée, jusques à ce que les 2 matrices aient été inversées, ou jusques à ce que cette opération aient été faite un certain nombre de fois
- ▶ cette opération introduit donc un nouveau paramètre dans le modèle
- ▶ on l'a simplement appelé n
- ▶ note : c'est souvent **Y2** qui pose problème, d'où une perte de temps, puisque l'inversion de **Y1** a déjà été faite, inutilement

Prony – Résumé



Prony – Code matlab/octave – 1/3

```
function [aa, ff, mydsp] = myprony(xx, pp, nco, fe, rec)
tsig = length(xx);
df=0.9765625; %%% la dsp est calculée tous les df Hz
ff=-fe/2 :df :fe/2;
strt=0; %%% échantillon de départ
count=0;
done=0;
while (done<1) %%% début de boucle principale
    ok1=0;
    ok2=0;
    %% alpha et f
    yy=[];
    for (ii=1 :pp)
        yy = [yy xx(pp+ii-1+strt :-1 :ii+strt)'];
    end;
    yy=yy';
    y1 = xx(pp+1+strt :2*pp+strt)';
    [invaa,condit]=inv(yy);
    if (condit>1.52721e-17) %%% conditionnement Y1
        ok1=1;
        aa = -invaa*y1;
        aa = [1 aa]';
        racines=roots(aa);
        alpha = log10(abs(racines));
        frequ = 1/2/pi*atan2(imag(racines),real(racines));
        mydsp(frequ, alpha);
        done=1;
    else
        count=count+1;
        if (count>100)
            done=1;
        end;
    end;
end;
```

Prony – Code matlab/octave – 2/3

```
%%%% rho et theta
zz=[];
for (ii=0 :pp-1)
    zz = [zz racines.^ii'];
end;
zz=zz';
y2 = xx(1 :pp)';
[invzz,condit]=inv(zz);
if (condit>1.52721e-17) %%% conditionnement Y2
    ok2=1;
end;
end;
if (ok1==1 & ok2==1)
    hh = invzz*y2;
    rho = abs(hh);
    theta = atan2(imag(hh),real(hh));
    done=1;
else
    fprintf(1,'pc ') ;fflush(1);
    count=count+1;
    if (count==nco)
        done=2;
    end;
    strt=round(rand(1,1)*(tsig-2*pp)); %%% choix d'un nouveau point de départ
end;
```

Prony – Code matlab/octave – 3/3

```
end; %%% fin de boucle principale
ff=ff/fe; %%% densité spectrale de puissance
if (done==1)
    if (rec==0)
        %%% DSP Prony
        sff=zeros(1,length(ff));za=exp(-j*2*pi*ff);zb=conj(za)
        for ii=1 :pp
            sff = sff + hh(ii) *( 1./(1 - racines(ii)*za ) - 1./(1 - ( conj(racines(ii))*zb ).^(-1) ) );
        end;
        mydsp=abs(sff).^2/fe;
    else
        %%% spectre de raies
        mydsp=zeros(length(ff),1)+1e-10;
        for ii=1 :pp
            [mini, posi] = min(abs(ff-frequ(ii)));mydsp(posi) = abs(hh(ii));
            [mini, posi] = min(abs(ff+frequ(ii)));mydsp(posi) = abs(hh(ii));
        end;
    end;
else
    fprintf(1,'unable to comply ');fflush(1);
    mydsp=zeros(1,length(ff))+1e-15;
    mydsp(round(length(ff)/2))=1e-5;
    aa=[];
end;
ff=ff*fe;
```

Prony – Premiers tests

On va faire quelques tests, avec des signaux (sons) simples pour vérifier qu'on obtient bien des densités spectrales de puissance intéressantes. Les 4 tests qu'on considère ici sont :

- ▶ on a une sinusoïde pure \Rightarrow est-ce qu'on arrive à la mettre en évidence ?
- ▶ on a deux sinusoïdes proches en fréquence \Rightarrow est-ce qu'on arrive à les discriminer ?
- ▶ on a un signal compliqué : une somme de sinusoïdes harmoniques \Rightarrow est-ce qu'on arrive à obtenir une enveloppe spectrale, modélisant ce signal, et/ou les n partiels ?
- ▶ on a un signal compliqué : un bruit rose \Rightarrow est-ce qu'on arrive à obtenir une enveloppe spectrale, modélisant ce signal ?

Prony – Une sinusoïde

myprony-matlab.m

Paramètres de l'exemple :

- ▶ $f_e = 32000$ Hz
- ▶ signal : une sinusoïde présente
 - ▶ de fréquence 440 Hz
 - ▶ et de phase aléatoire, tirée entre 0 et 2π
 - ▶ de taille 1280 échantillons (40 ms)
- ▶ $p = 40$ et $n = 10$

Prony – Une sinusoïde

Attention :

- ▶ on a à présent ces 2 paramètres, p et n , à faire varier
 - ▶ difficultés similaires à ce qu'on avait avec MUSIC
- ▶ on est obligé d'ajouter un peu de bruit blanc, pour assurer mieux la stabilité de l'algorithme
 - ▶ il faut noter que c'est surtout l'inversion de la matrice des z (Y2) qui pose problème
 - ▶ la variance de ce bruit est petite (mais il est visible sur la forme d'onde) : $\sigma^2 = 1e^{-2}$
- ▶ il faut voir aussi que les signaux qu'on considère dans nos tests ne sont pas précisément des sinus ou des exponentielles amortis

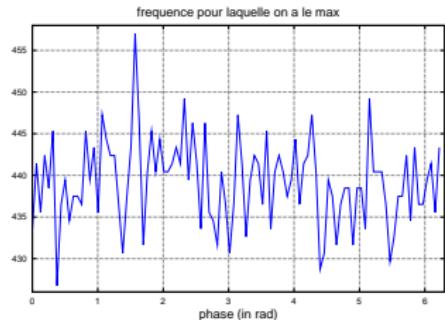
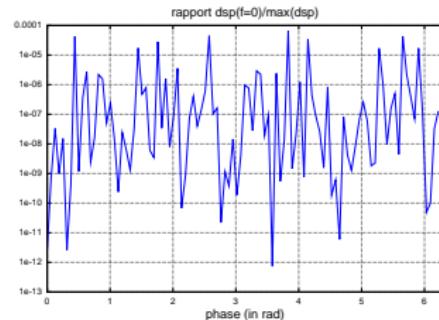
Prony – Une sinusoïde

scriptphaseprony.m

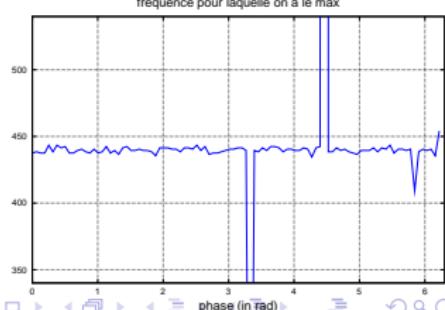
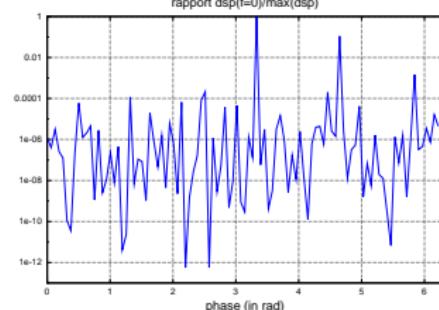
- ▶ on regarde ce qui se passe en fonction de la phase de la sinusoïde
- ▶ on mesure le rapport entre l'amplitude du spectre de raies en $f = 0$ et son amplitude maximum (sensée se trouver à $\pm 440 \text{ Hz}$)
- ▶ on regarde où se trouve en réalité l'amplitude maximum du spectre de raies
- ▶ ce pour différents ordres : $p = 2, p = 3, \dots$ et pour différents $n : n = 1, n = 2, n = 3, \dots$

Prony – Une sinusoïde

$p = 50, n = 10 :$



$p = 100, n = 5 :$



Prony – Une sinusoïde

- ▶ nombre de fois sur 100 où fréq. du max. trouvée entre 400 Hz et 480 Hz
- ▶ nombre entre () = nombre d'échecs dus au mauvais conditionnement

n	1	2	5	10	15	20	25
$p = 2$	0	—	—	—	—	—	—
$p = 3$	0	—	—	—	—	—	—
$p = 10$	7(1)	—	—	—	—	—	—
$p = 30$	80(11)	87	91	92	93	91	87
$p = 35$	78(17)	99	98	98	96	95	98
$p = 40$	81(18)	96(3)	98	100	—	—	—
$p = 50$	76(24)	90(8)	100	—	—	—	—
$p = 100$	55(37)	85(15)	98(1)	100	—	—	—
$p = 150$	47(52)	67(32)	94(6)	100	—	—	—
$p = 200$	32(68)	59(39)	87(13)	99(1)	98(2)	100	100

- ▶ — indique que la mesure est inutile ; — mesure pas très utile

Prony – Une sinusoïde

- ▶ on passe par un optimum en p ; au-delà, c'est des problèmes de conditionnement de matrices qui surviennent
- ▶ il faut noter quand même à quel point on peut descendre en-dessous de 1280 échantillons de signal!
 - ▶ avec 80 ou même 60 échant. ($p = 40$ ou $p = 30$), on arrive déjà à souvent s'en sortir
 - ▶ 60 échant., c'est moins d'1 période du sinus de plus basse fréquence

Prony – Une sinusoïde

- ▶ pour $p = 100$, on pourrait penser qu'à la place de 85 on devrait avoir $55 + (37 - 15) = 77$, mais il ne faut pas oublier que du bruit est ajouté, qui est re-tiré aléatoirement pour chaque trame de signal de 1280 échantillons ; et de plus les portions de signal considérées en cas de mauvais conditionnement sont elles-mêmes prises au hasard
- ▶ note 1 : comme on tire parfois les sous-trames aléatoirement dans la trame de 1280 échantillons, la phase indiquée sur les figures précédentes n'a plus forcément de sens
- ▶ note 2 : pour $p = 2$, $n = 1$, s'il n'y a pas de bruit additif, on obtient 100 % de succès

Prony – Intérêt

- ▶ est-ce que l'algorithme de Prony présente le même intérêt que les autres méthodes en termes de haute-résolution ?
- ▶ on refait les mêmes tests qu'avec les méthodes AR, Pisarenko, MUSIC et la FFT
 - ▶ 2 sinusoïdes proches, séparées de 26 Hz (440 Hz et 466 Hz) et de même amplitude, etc. : voir transparent suivant
- ▶ et on fait varier les ordres p et n
 - ▶ on part d'ordres aussi petits que possible

Prony – Intérêt

- ▶ longueur du signal : 1280 échantillons (une seule trame de 40 ms)
- ▶ sinusoïdes de fréquences 440 Hz et 466 Hz
- ▶ la DSP est calculée sur 32768 points
- ▶ le résultat obtenu dépend de la phase respective des 2 sinus
- ▶ il dépend aussi des ordres p et n du modèle de Prony

Prony – Intérêt

- ▶ la méthode de détection est un peu différente
 - ▶ comme avant, on détecte les max et min locaux entre 400 et 506 Hz, puis on vérifie qu'il y a bien 2 max locaux, et 1 min local, se situant entre les 2 max locaux
 - ▶ mais il y a des min locaux supplémentaires : on ne garde que celui/ceux qui sont entre les 2 premiers max locaux
 - ▶ mais il y a des max locaux supplémentaires : on ne garde que ceux dont l'amplitude est supérieure au max de la DSP divisé par 1000 (seuil supplémentaire)
 - ▶ note : à tester chez les autres analyseurs
- ▶ Prony comme Pisarenko donne des raies : on peut donc faire comme dans *scriptdoublonpisarenko2.m* : détection des 2 plus grandes raies, qui doivent se trouver dans la bande 400-506 Hz

Prony – Intérêt

► voir :

scriptdoublonprony.m et *scriptdoublonprony2.m*

Prony – Intérêt

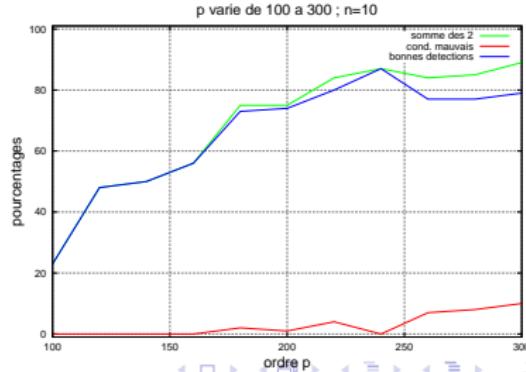
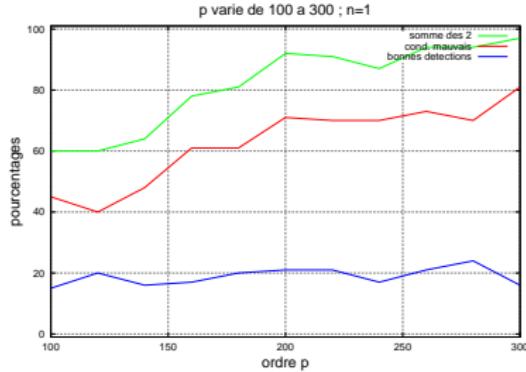
- ▶ En ce qui concerne *scriptdoublonprony.m*, on a (variance du bruit $\sigma^2 = 1e - 2$) :

n	1	2	5	10	15	20
$p = 100$	1.9(41.9)	2.2(16.8)	3(0)	8(0)	0(0)	4(0)
$p = 200$	13.8(67.7)	27.5(43.4)	55(5)	61(2)	49(0)	48(0)
$p = 400$	10(89)	29(70)	58(40)	76(22)	68(15)	92(3)

- ▶ une exploration plus exhaustive serait requise (il faut noter que tout ceci est fort coûteux en temps de calcul)
- ▶ variance du bruit trop grande ? ; ou ordre p pas assez ajusté ?
- ▶ à partir de $n = 5$ et pour $p > 200$: nombre d'essais par case = 100

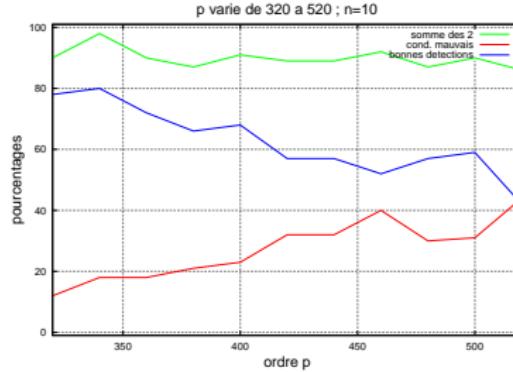
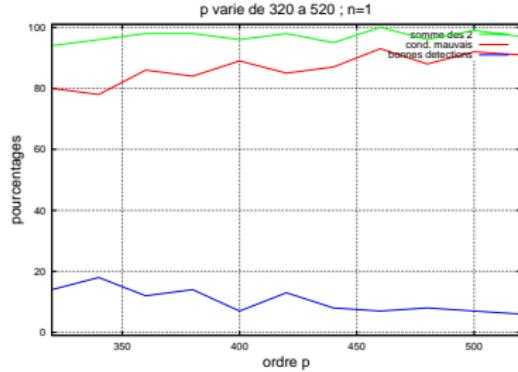
Prony – Intérêt

En ce qui concerne *scriptdoublonprony2.m* (variance du bruit $\sigma^2 = 1e - 3$), p varie entre 100 et 300 avec un pas de 20, et on examine ce qu'on obtient pour $n = 1$ et $n = 10$. Le nombre d'essais pour chaque point est 100 (pour les autres méthodes, c'est 625).



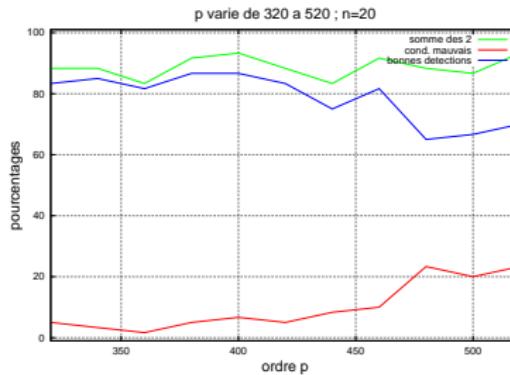
Prony – Intérêt

p varie entre 320 et 520 avec un pas de 20, et on examine ce qu'on obtient pour $n = 1$ et $n = 10$. Le nombre d'essais pour chaque point est 100 (pour les autres méthodes, c'est 625).



Prony – Intérêt

p varie entre 320 et 520 avec un pas de 20, et on examine ce qu'on obtient pour $n = 20$. Le nombre d'essais pour chaque point est 100 (pour les autres méthodes, c'est 625).



Prony – Intérêt

- ▶ encore une fois, on fait mieux qu'avec la FFT, même si on n'atteint pas 100 % de bonnes détections
- ▶ comme pour MUSIC, c'est difficile d'optimiser les 2 paramètres
- ▶ il faut noter qu'il faut quand même prendre p grand (l'ordre le plus grand possible étant 640)
- ▶ il est normal que les performances tendent à se dégrader quand on a des ordres très grands : en effet, dans ce cas, le nombre de sous-trames possibles se réduit énormément

Prony – Enveloppe

- ▶ ON CONSIDÈRE LE SPECTRE DE RAIRES, PUIS LA DSP
- ▶ on a un signal harmonique (composé d'une somme de partiels harmoniques). Le signal considéré :
 - ▶ fréquence fondamentale : $f_0 = 220 \text{ Hz}$
 - ▶ nombre de partiels : 20
 - ▶ amplitudes des partiels : elles suivent une gaussienne centrée en $f_{max} = 2500 \text{ Hz}$ et d'écart-type 1500
 - ▶ voir transparent suivant
 - ▶ phases des partiels : tirées aléatoirement entre 0 et 2π
 - ▶ la longueur du signal est toujours 1280 échantillons, et la fréquence d'échantillonnage 32000 Hz
 - ▶ bruit additif : $\sigma^2 = 1e^{-2}$

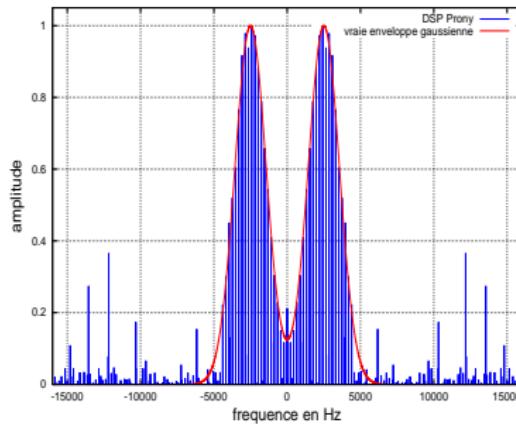
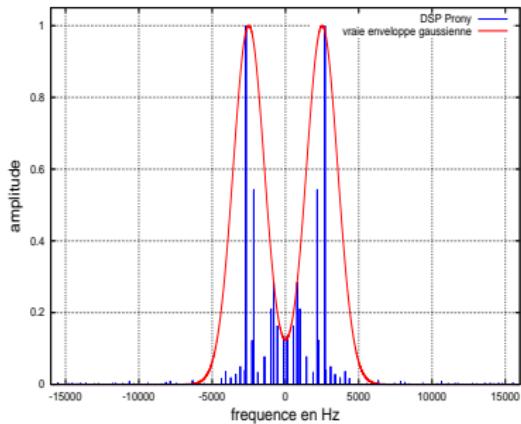
Prony – Enveloppe

► voir :

scriptenvelopprony.m

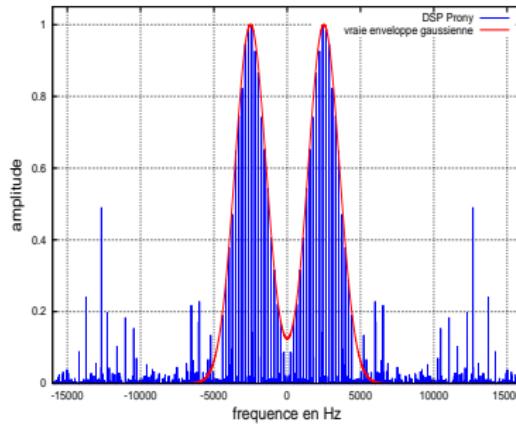
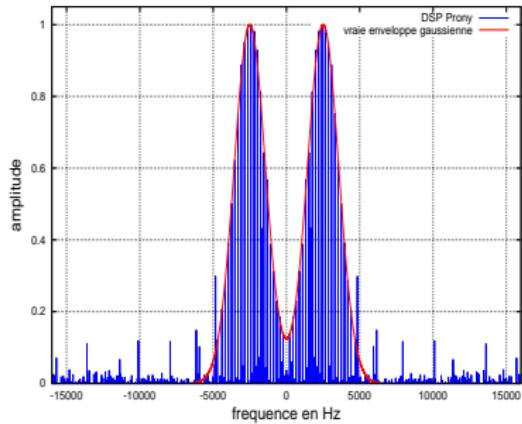
Prony – Enveloppe

Spectres obtenus avec le signal précédent, $p = 100$ et $p = 200$,
 $n = 10$, norme 1 :



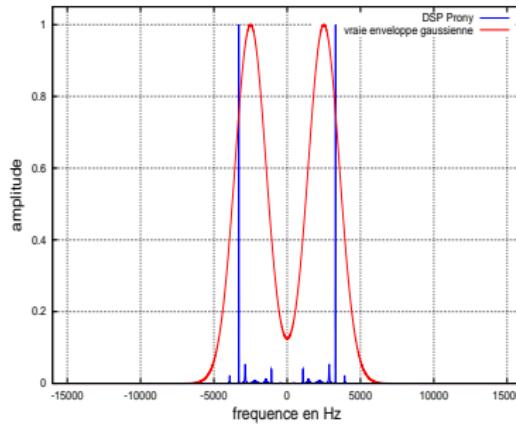
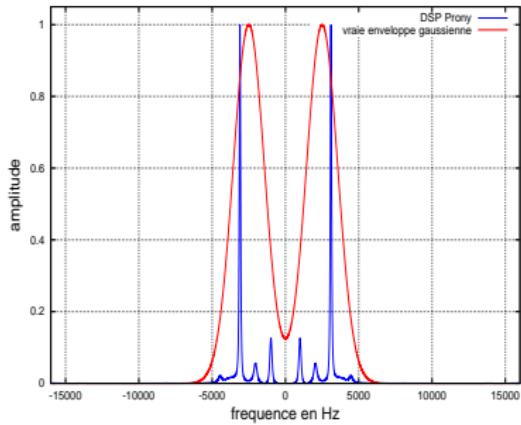
Prony – Enveloppe

Spectres obtenus avec le signal précédent, $p = 300$ et $p = 400$,
 $n = 10$, norme 1 :



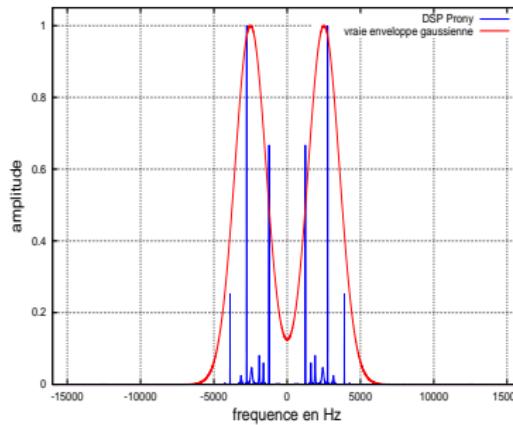
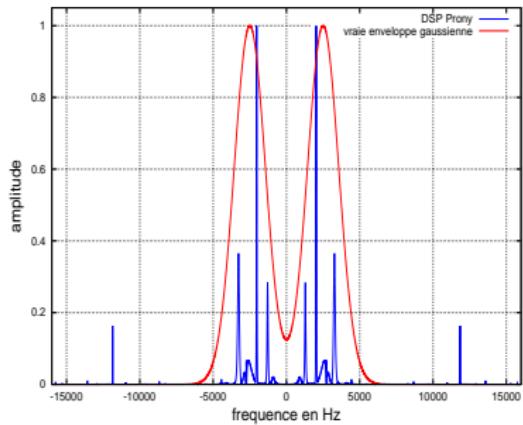
Prony – Enveloppe

DSP obtenues avec le signal précédent, $p = 25$ et $p = 50$, $n = 10$, norme 1 :



Prony – Enveloppe

DSP obtenues avec le signal précédent, $p = 75$ et $p = 100$,
 $n = 10$, norme 1 :

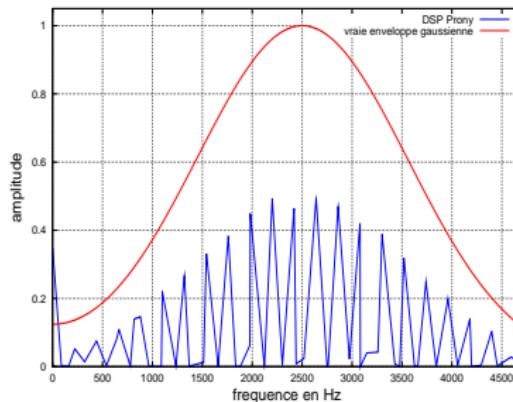
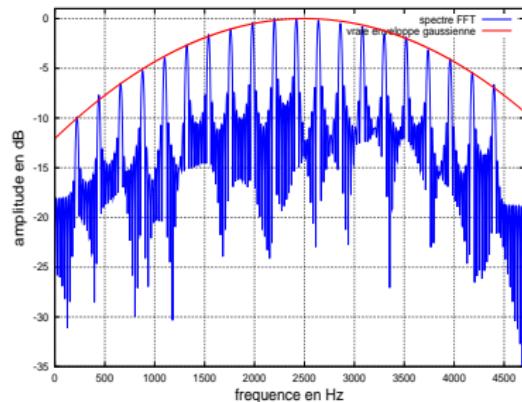


Prony – Enveloppe

- ▶ spectre de raies :
 - ▶ la majorité des pics tombent dans la bonne bande de fréquences
 - ▶ par contre, il faudrait traiter les spectres de raies, pour espérer pouvoir en tirer quelque information
- ▶ DSP :
 - ▶ même pour de petits ordres, elle suit déjà trop les partiels
 - ▶ donc, assez curieusement, il vaudrait peut-être mieux interpôler linéairement entre les pics du spectre de raies, comme on fait pour Pisarenko (on va appeler ça DSP Prony-Interpolation ci-dessous)

Prony – Enveloppe

Exemple de spectre FFT et DSP Prony-Interpolation ($p = 400$, $n = 10$, norme 3) obtenus avec le signal précédent :



⇒ on voit qu'il faut normaliser la DSP, et que l'ordre p est trop grand



Prony – Enveloppe

Mesures :

- ▶ différence entre 0 et 4700 Hz pour chaque fréquence entre DSP obtenue et vraie enveloppe gaussienne \Rightarrow la somme normalisée donne la distance entre les 2
- ▶ on fait cette mesure un grand nombre de fois : le résultat dépend des phases respectives des partiels
- ▶ on la fait pour différents ordres p , avec $n = 10$
- ▶ ON PREND LA MÉDIANE AU LIEU DE LA MOYENNE, à cause des cas de conditionnement défaillant et des pics parasites très grands qui perturbent la normalisation

Prony – Enveloppe – Norme 1

p	n	N	min	max	median	σ
100	10	100	0.164119	1.026290	0.663938	0.173741
130	10	100	0.250940	0.997589	0.732442	0.197548
140	10	100	0.024274	10.000000	0.188152	1.016812
150	10	100	0.024173	0.863406	0.043293	0.209667
160	10	100	0.052163	10.000000	0.085646	1.396175
170	10	100	0.074430	10.000000	0.120730	1.004888
200	10	100	0.143746	1.000281	0.209812	0.248081
300	10	100	0.443118	1.000281	0.503097	0.175912

Note : 4 % d'erreur, on n'a pas obtenu mieux jusque à présent !

Prony – Enveloppe – Norme 2

p	n	N	min	max	median	σ
100	10	100	0.214329	1.232194	0.651686	0.229367
130	10	100	0.213202	1.763408	0.720972	0.281852
140	10	100	0.023555	0.937788	0.112231	0.167110
150	10	100	0.024600	0.372210	0.044383	0.054721
160	10	100	0.048782	0.502246	0.100173	0.072116
170	10	100	0.057943	0.689029	0.149224	0.080890
200	10	100	0.214801	10.000000	0.273656	1.368339
300	10	100	0.450905	10.000000	0.505987	2.729707

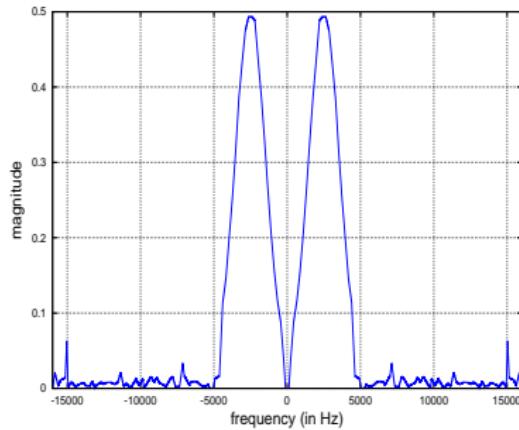
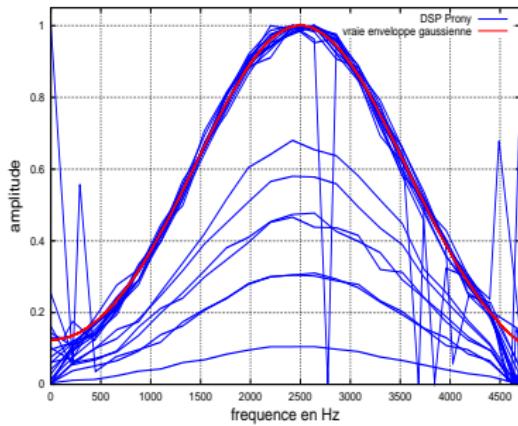
On peut se contenter de la norme 1

Prony – Enveloppe

- ▶ la qualité des enveloppes spectrales obtenues est équivalente à ou même bien meilleure que celle des enveloppes spectrales obtenues avec les modèles AR et les autres méthodes de projection
- ▶ il faut prendre la médiane pour limiter les effets des cas aberrants, qui viennent de ce que l'algorithme ne parvient pas toujours à converger ou à bien converger
- ▶ on a obtenu pour une petite plage de p des DSP d'excellente qualité
- ▶ il faut quand même noter qu'il y a des pics parasites sur les côtés, d'amplitudes pas toujours négligeables

Prony – Enveloppe

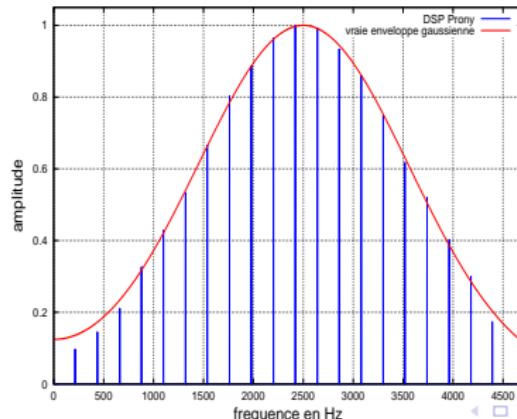
20 DSP Prony-Interpolation ($p = 150$, $n = 10$, norme 1), et un exemple avec norme 3 :



Prony – Enveloppe – Norme 1

Est-ce qu'en augmentant suffisamment les ordres p et n on arrive à récupérer les 20 partiels ?

Oui ! Ainsi, pour $p = 150$, $n = 10$ et norme 1, on obtient par exemple cet excellent spectre de raies :



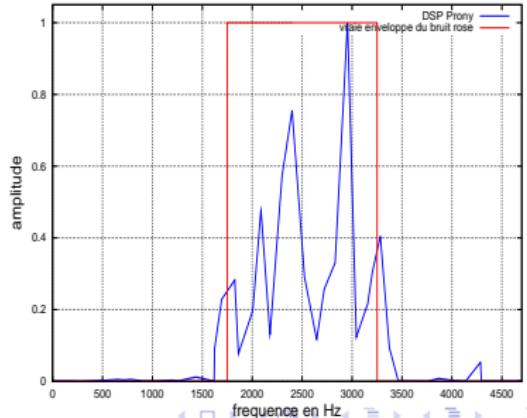
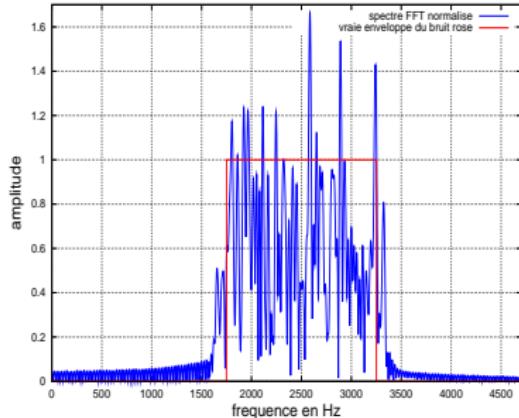
Prony – Bruit rose

Le bruit rose considéré :

- ▶ fréquence centrale du bruit rose : $f_c = 2500 \text{ Hz}$
- ▶ largeur de la bande : 1500 Hz
- ▶ l'amplitude du bruit vaut 1 dans la bande, et 0 donc en-dehors
 - ▶ voir transparent suivant
- ▶ on fait d'abord un bruit blanc, qu'on filtre après coup
- ▶ la longueur du signal est toujours 1280 échantillons, et la fréquence d'échantillonnage 32000 Hz

Prony – Bruit rose

Exemple de spectre FFT et de DSP Prony-Interpolation ($p = 100$, norme 1) obtenus avec le signal précédent (la DSP Prony est trop piquée, même pour de petits ordres, pour qu'on l'utilise) :



Prony – Bruit rose

Note : la normalisation du spectre FFT est parfaitement contrôlée
(considérations énergétiques concernant les spectres continus)

Mesures :

- ▶ on calcule entre 0 Hz et 4700 Hz la différence pour chaque fréquence entre la DSP normalisée obtenue et la vraie enveloppe du bruit rose \Rightarrow la somme normalisée donne la distance entre les deux
- ▶ on fait cette mesure un grand nombre de fois : le résultat dépend du tirage initial du bruit blanc
- ▶ on fait cette mesure pour différents ordres p , avec $n = 10$

Prony – Bruit rose

► voir :

scriptpinkprony.m

Prony – Bruit rose – Norme 1

p	n	N	min	max	median	σ
50	10	100	0.324899	1.500332	0.713112	0.205452
74	10	100	0.310644	1.254209	0.691123	0.170635
100	10	100	0.368251	1.139833	0.690649	0.136412
124	10	100	0.463710	1.005941	0.701076	0.108555
150	10	100	0.421613	1.133169	0.717128	0.114975
200	10	100	0.515200	1.077867	0.740450	0.099262
250	10	100	0.546904	10.00000	0.766925	2.205320

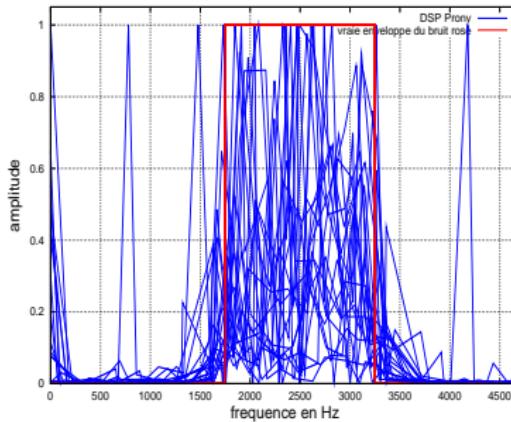
Prony – Bruit rose

- ▶ il y a un souci avec la normalisation de la DSP (comme pour Pisarenko et MUSIC)
- ▶ la normalisation utilisée pour obtenir les précédents résultats consistait seulement à mettre le maximum de la DSP à 1 (*en pratique, par FFT, on a un peu de mal à estimer le maximum de l'enveloppe réelle*) [norme 1]
- ▶ sur la figure suivante, on voit que ça amène comme précédemment à sous-estimer l'amplitude de l'enveloppe
- ▶ on pourrait normaliser la surface de la DSP entre 0 Hz et 4700 Hz pour qu'elle corresponde à celle de la vraie DSP (*en pratique, on n'a pas facilement accès à cette information*) [norme 2]

Prony – Norme 1

20 DSP Prony-Interpolation :

$p = 150, n = 10$:



Prony – Norme 2

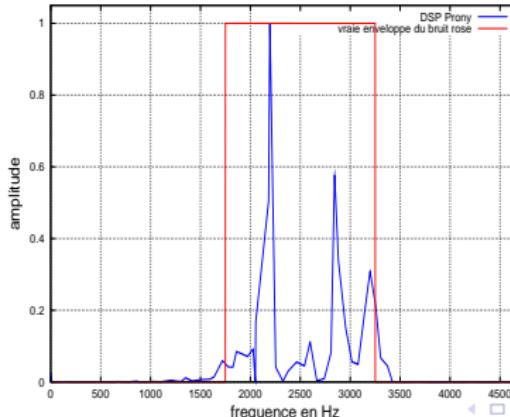
Utiliser la deuxième normalisation n'est pas très utile :

p	n	N	min	max	median	σ
50	10	100	0.344073	1.395656	0.715292	0.191922
74	10	100	0.331948	1.571483	0.698992	0.232150
100	10	100	0.343391	1.309998	0.607952	0.175930
124	10	100	0.362572	10.00000	0.675198	0.952885
150	10	100	0.314501	1.244197	0.674807	0.162957
200	10	100	0.370675	10.00000	0.723041	1.319344
250	10	100	0.448387	10.00000	0.692622	1.313859

Prony – Bruit rose – Norme 1

Et si on prend un ordre p extrêmement grand, qu'est-ce qui se passe ?

Pour $p = 500$ ($n = 10$, Prony-Interpolation), on obtient par exemple :



Résumé des résultats

	L.-Durb.	Burg	Marple	Pisarenko	MUSIC	Prony-H	FFT
1 sin	$p \geq 4$	$p \geq 2$	$3 \leq p \leq 10$ $\sigma^2 = 1e^{-4}$ $4 \leq p \leq 40$ $\sigma^2 = 1e^{-3}$	$p \geq 1$	$p = 1 \Rightarrow M \geq 110$ $p \geq 2 \Rightarrow M \geq p + 2$	$p \geq 40$ $n \geq 10$	oui
2 sin	$p=250$ 63% $p=265$ 88% $p=300$ 100%	$p=4$ 99.68% $p=5$ 100% $p=6$ 97% $p=7$ 7%	$p=90$ 16% $p=100$ 83% $p=110$ 93% $p=120$ 86%	m. autre $p=40$ 33% $p=80$ 60% $p=100$ 77% $p=150$ 94%	$p = 4$ $M_{=90}$ 23% $M_{=100}$ 73% $M_{=110}$ 98.6% $M_{=120}$ 68%	m. autre $p \geq 180$ $n \geq 10$ $\leq 87\%$	B 32% Hn 40% R 64% méth. autre
env	$e_1 : 0.318$ $\bar{e}_2 : 0.305$	pas très efficace	$e_1 : 0.337$ $\bar{e}_2 : 0.310$	$e_3 : 0.392$	$e_1 : 0.527$ $\bar{e}_2 : 0.462$	$e_1 : 0.043$ $\bar{e}_2 : 0.044$	dét. pics à faire
20 sin	oui $p=200$	oui $p=100$	oui $p=140$	dur	oui $pM_{=50,150}$	oui $pn_{=150,10}$	oui
rose	$\bar{e}_1 : 0.568$ $\bar{e}_2 : 0.367$ $e_3 : 0.378$	$\bar{e}_1 : 0.708$ $\bar{e}_2 : 0.456$ $e_3 : 0.493$	$\bar{e}_1 : 0.520$ $\bar{e}_2 : 0.344$	$\bar{e}_1 : 0.99$	$\bar{e}_1 : 0.746$ $\bar{e}_2 : 0.527$	$\bar{e}_1 : 0.690$ $\bar{e}_2 : 0.608$	$e \simeq 0.5$

Prony – Les amplitudes (norme 3)

- ▶ *myprony_matlab.m* : amplitude du sinus : 1, ; on obtient un pic d'amplitude de l'ordre de 0.5 (spectre de raies) ou 100 (DSP)
- ▶ *scriptdoublonprony.m* : amplitude de chaque sinus : 1 ; on obtient des pics de grandes amplitudes (ordre de grandeur : 50 et 600)
- ▶ *scriptdoublonprony2.m* : l'amplitude de chaque sinus vaut 1, et on obtient des pics d'amplitude de l'ordre de 0.5
- ▶ *scriptenvelopprony.m* : l'amplitude du plus grand partiel est 1, et on obtient un maximum dont l'amplitude est de l'ordre de 0.5 (spectre de raies) ou 16000 (DSP)
- ▶ *scriptpinkprony.m* : amplitude dans la bande du bruit rose : 1 ; on obtient un spectre de raies ou une DSP trop petits (de l'ordre de 0.2)

Prony – Le code matlab/octave

- ▶ j'ai mis le code de *myprony_matlab.m* sur mon site :
[http://www.metz.supelec.fr/metz/personnel/
rossignol/myprony_matlab.m](http://www.metz.supelec.fr/metz/personnel/rossignol/myprony_matlab.m)
- ▶ si vous le souhaitez, vous pouvez faire profiter votre rapport des résultats que vous obtenez avec ces méthodes

Prony – Le code en C

- ▶ dans une perspective d'utilisation de ces méthodes d'analyse spectrale en temps réel, il faudrait traduire ce code en C
- ▶ si on examine le code matlab/octave, on voit qu'il y a des complications ; on a besoin :
 - ▶ des opérations de manipulation des matrices : inversion, avec mesure de conditionnement, etc.
 - ▶ de l'extraction des racines d'un polynôme
- ▶ voir des bibliothèques pré-existantes pour le calcul matriciel, ou alors fouiner dans les Numerical Recipes
- ▶ bien sûr, il faudrait aussi considérer tous les problèmes liés à l'acquisition des signaux

- ▶ Introduction
- ▶ Modèles AR
- ▶ Pisarenko
- ▶ MUSIC/ESPRIT
- ▶ Prony
- ▶ Liens avec la localisation de sources

Liens avec la localisation de sources

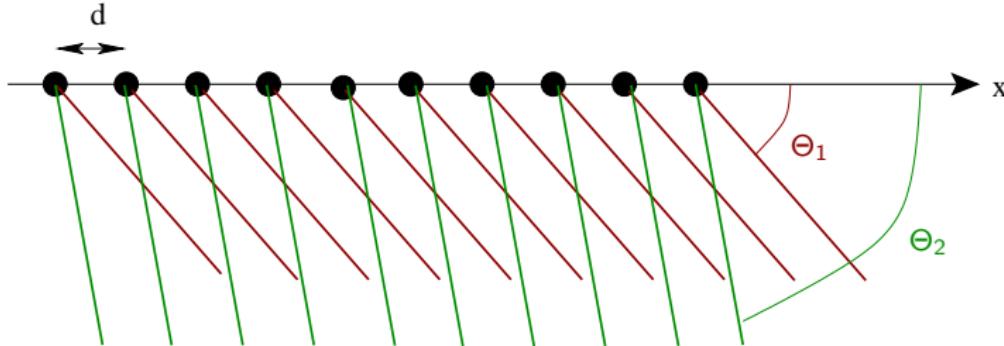
- ▶ dans le cas d'un réseau d'antennes linéairement et régulièrement espacées, la conversion est quasi directe
- ▶ il suffit de considérer le signal échantillonné dans le domaine temporel, pour lequel on a :

$$s(t) = \sum_{i=1}^m \cos(2\pi f_i t) \sum_{j=0}^{N-1} \delta(t - j/f_e)$$

où m est le nombre de raies, f_i leurs fréquences, N le nombre d'échantillons considérés, et f_e la fréquence d'échantillonnage (on oublie les amplitudes et les phases, pour simplifier)

Liens avec la localisation de sources

- ▶ puis de considérer maintenant le cas du réseau d'antennes ci-dessous :



où il y a N antennes séparées de d , et où par exemple on reçoit un signal de deux directions Θ_1 et Θ_2 ($m = 2$), chacun étant émis à la même longueur d'onde λ

Liens avec la localisation de sources

- ▶ alors, en remplaçant le temps t par la position x et la fréquence << temporelle >> f_i par la fréquence spatiale Θ_i , on peut écrire :

$$s(x) = \sum_{i=1}^m \cos\left(2\pi \frac{\cos(\Theta_i)}{\lambda} x\right) \sum_{j=0}^{N-1} \delta(x - jd)$$

où maintenant N est le nombre d'antennes et m le nombre de sources

- ▶ dès lors, les DSP, par exemple des modèles AR, se réécrivent directement :

$$P_{ss}(\Theta) = \frac{d\sigma^2}{\left| 1 + \sum_{k=1}^p a_k \exp(-j2\pi kd \cos(\Theta)/\lambda) \right|^2}$$

Liens avec la localisation de sources

- ▶ bien sûr, pour $s(x)$, il faut réintroduire la dimension temporelle : $s(x)$ est observé tous les t_j , pour $j = [1 \dots J]$, et J est le nombre d'observations
- ▶ ces instants t_j n'ont plus tout à fait la même signification que les t de l'analyse spectrale
- ▶ les autocorrélations de $s(x)$ sont moyennées sur toutes les observations J
- ▶ note : cette autocorrélation moyennée est utilisée pour certaines méthodes AR et pour MUSIC
- ▶ MUSIC se réécrit facilement dans le cas de réseaux d'antennes non-linéaires ; ce n'est pas le cas pour les méthodes AR

- ▶ Introduction
- ▶ Modèles AR
- ▶ Pisarenko
- ▶ MUSIC/ESPRIT
- ▶ Prony
- ▶ Liens avec la localisation de sources
- ▶ Performances, choix de l'ordre, modèles MA et ARMA

Autres performances des méthodes

- ▶ on a déjà vu un transparent indiquant une partie des autres tests qu'il faudrait faire
- ▶ on ne va considérer qu'une toute petite partie de ceux-ci : ce qui concerne la résistance des méthodes d'analyse spectrale au bruit
- ▶ on va reprendre le signal composé d'une seule sinusoïde, auquel on ajoute du bruit de variance variable
- ▶ note : on tombe sur des problèmes où il y a plusieurs paramètres qui varient ; ici, c'est p à optimiser en fonction de σ^2
- ▶ on ne va donc pas pouvoir être exhaustif, ni de près ni de loin

Autres performances des méthodes

- ▶ voir *scriptsnrfft.m*
- ▶ on donne le nombre de fois sur 100 où le maximum du spectre tombe entre 400 Hz et 480 Hz, pour des SNR allant de -20 dB à -9 dB

σ^2	60	50	30	15	7.5	3.75
Rectangulaire	32	46	87	100	100	100
Blackman	20	32	47	92	100	100
Hanning	18	28	65	89	100	100
Hamming	27	38	65	95	100	100

- ▶ 80 Hz couvre 0.5% du spectre : dès lors, quand on trouve plus de 0.5% de bonnes détections, à la statistique du phénomène prêt, on peut dire qu'on commence à détecter la sinusoïde

Autres performances des méthodes

- ▶ voir *scriptsnrlevinson.m*
- ▶ on donne le nombre de fois sur 100 où le maximum du spectre tombe entre 400 Hz et 480 Hz

σ^2	60	50	30	15	7.5	3.75
$p = 50$	12	22	34	76	94	100
$p = 100$	14	16	52	92	100	100
$p = 150$	26	37	67	98	100	100
$p = 250$	31	28	62	98	100	100

Autres performances des méthodes

- ▶ voir *scriptsnrburg.m*
- ▶ on donne le nombre de fois sur 100 où le maximum du spectre tombe entre 400 Hz et 480 Hz

σ^2	60	50	30	15	7.5	3.75
$p = 50$	11	19	30	78	96	99
$p = 100$	22	21	57	96	99	100
$p = 150$	28	40	60	97	100	100
$p = 250$	20	37	67	97	100	100

Autres performances des méthodes

- ▶ voir *scriptsnrmarple.m*
- ▶ on donne le nombre de fois sur 100 où le maximum du spectre tombe entre 400 Hz et 480 Hz

σ^2	60	50	30	15	7.5	3.75
$p = 50$	18	19	30	75	95	100
$p = 100$	19	25	54	92	100	100
$p = 150$	29	31	68	99	100	100
$p = 250$	31	38	72	97	100	100

Autres performances des méthodes

- ▶ voir *scriptsnrpisarenko.m*
- ▶ on donne le nombre de fois sur 100 où le maximum du spectre tombe entre 400 Hz et 480 Hz

σ^2	60	50	30	15	7.5	3.75
$p = 50$	4	5	9	24	48	70
$p = 100$	14	16	23	52	82	91
$p = 150$	7	11	21	44	75	84
$p = 250$	15	25	39	69	90	96

Autres performances des méthodes

- ▶ voir *scriptsnrnmusic.m*
- ▶ on donne le nombre de fois sur 100 où le maximum du spectre tombe entre 400 Hz et 480 Hz

	σ^2	60	50	30	15	7.5	3.75
$p = 50$	$M = 100$	18	18	39	71	94	98
$p = 100$	$M = 150$	8	21	37	71	92	100
$p = 150$	$M = 200$	12	16	46	63	89	100
$p = 250$	$M = 300$	10	20	35	62	87	95

Autres performances des méthodes

- ▶ voir *scriptsnrprony.m*
- ▶ le SNR varie entre -14 dB et 17 dB
- ▶ on donne le nombre de fois sur 100 où le maximum du spectre tombe entre 400 Hz et 480 Hz

	σ^2	15	7.5	3.75	1	0.1	$1e^{-2}$
$p = 50$	$n = 10$	0	3	2	1	33	61
$p = 100$	$n = 10$	1	0	0	0	13	64
$p = 150$	$n = 10$	0	0	0	1	20	60
$p = 250$	$n = 10$	0	0	0	0	20	58

Autres performances des méthodes

- ▶ Levinson-Durbin, Burg et Marple se comportent au moins aussi bien que la FFT
- ▶ les mesures faites ne permettent pas de les classer plus finement
 - ▶ par exemple, à -20 dB, on est au mieux aux alentours de 30 % de bonnes détections (BD) pour toutes les méthodes d'analyse spectrale
 - ▶ le passage à plus de 75 % de BD a lieu entre $\sigma^2 = 30$ et $\sigma^2 = 15$
- ▶ Pisarenko semble moins robuste que les 3 méthodes AR : à -20 dB, on a 15 % de BD, et on passe à plus de 75 % de BD entre $\sigma^2 = 15$ et $\sigma^2 = 7.5$; l'ordre ne semble pas aider

Autres performances des méthodes

- ▶ MUSIC semble moins robuste que les 3 méthodes AR et se comporte quasi de la même façon Pisarenko : à -20 dB, on a 15 % de BD environ, et on passe à plus de 75 % de BD entre $\sigma^2 = 15$ et $\sigma^2 = 7.5$; l'ajustement de M est à faire plus judicieusement en fonction de l'ordre
- ▶ Prony est beaucoup plus sensible au bruit que les autres méthodes ; même avec un SNR de 17 dB, on a encore moins de 75 % de bonnes détections ; augmenter l'ordre de semble pas aider (ajustement de n ?)

Sélection de l'ordre p

- ▶ dans la réalité, comme déjà abondamment mentionné, on ne sait pas quel ordre p choisir
- ▶ il y a un tas de critères sensés aider à décider de l'ordre à prendre (aucun ne marchant de façon complètement satisfaisante)
- ▶ il faut noter que lors de la recherche itérative des paramètres AR, on obtient à chaque itération k la variance du bruit blanc générateur $\hat{\sigma}_k^2$: on aura besoin de ces variances successives dans les critères qui suivent
- ▶ note : tous les critères qui suivent sont à minimiser selon k
- ▶ je vous laisserai les tester par vous-même

Sélection de l'ordre p

- ▶ [1.] le premier critère est FPE (Final Prediction Error) ; on peut calculer ce terme, où N est le nombre d'échantillons du signal, à chaque itération k :

$$FPE(k) = \frac{N + (k + 1)}{N - (k + 1)} \hat{\sigma}_k^2$$

- ▶ [2.] il y a ensuite AIC (Akaike Information Criterion) ; on peut calculer ce terme, à chaque itération k :

$$AIC(k) = \frac{2k}{N} + \log(\hat{\sigma}_k^2)$$

Sélection de l'ordre p

- ▶ [3.] il y a ensuite MDL (Minimum Description Length) ; on peut calculer ce terme, à chaque itération k :

$$MDL(k) = \frac{k \log N}{N} + \log(\hat{\sigma}_k^2)$$

- ▶ [4.] il y a ensuite CAT (Criterion Autoregressive Transfer) ; on peut calculer ce terme, à chaque itération k :

$$CAT(k) = \frac{1}{N} \sum_{i=1}^k \left(\frac{N-i}{N} \frac{1}{\hat{\sigma}_i^2} \right) - \frac{N-k}{N} \frac{1}{\hat{\sigma}_k^2}$$

Modèles MA

- ▶ le modèle est :

$$x_n = \sum_{l=0}^q b_l n_{n-l}$$

- ▶ il faut estimer les paramètres b_l
- ▶ cette expression donne directement la réponse impulsionnelle du filtre associé à la modélisation MA : $h_i = b_i \quad i \in [0, q]$
- ▶ la DSP d'un MA est fournie par l'expression :

$$S(f) = \left| \sum_{l=0}^q b_l \exp(-j2\pi lf/f_e) \right|^2$$

Les modèles MA avec les mains

- ▶ on va faire le lien entre :
 - ▶ la position des zéros de $B(z)$ avec :
$$B(z) = b_0 + b_1z^{-1} + b_2z^{-2} + \dots$$
 - ▶ les coefficients MA : $\{b_0, b_1, b_2, \dots\}$

- ▶ et la DSP obtenue : $S(\nu) = \left| \sum_{l=0}^q b_l \exp(-j2\pi l\nu) \right|^2$

- ▶ on va jouer avec *aveclesmains_ma.m*

Les modèles MA avec les mains

- ▶ essai 1 :
 - ▶ on a 2 zéros complexes conjugués \Rightarrow donc le signal est réel
 - ▶ de phase constante égale $\pm 50.4^\circ$; ça correspond donc à un creux de fréquence réduite $\nu = 50.4/360 = 0.14$ ou de fréquence $f = 4480 \text{ Hz}$ si $f_e = 32000 \text{ Hz}$
 - ▶ et on augmente leur amplitude progressivement
- ▶ essai 2 : on a 6 zéros :
 - ▶ $0.99 \exp(\pm j2\pi 0.05) \quad 0.98 \exp(\pm j2\pi 0.20) \quad 0.8 \exp(\pm j2\pi 0.3)$
 - ▶ les zéros sont complexes conjugués 2 à 2 \Rightarrow donc le signal est réel

Modèles MA

- ▶ l'estimation des paramètres peut se faire de différentes façons ; on ne va pas entrer dans les détails
- ▶ 1. Méthode des moments (voir RASS) ; on résoud un système non-linéaire de ce type (on peut tomber sur un minimum local) :

$$\begin{bmatrix} b_0 & \dots & b_q \\ b_1 & \ddots & 0 \\ \vdots & \ddots & \vdots \\ b_q & \dots & 0 \end{bmatrix} \begin{bmatrix} b_0 \\ \vdots \\ b_q \end{bmatrix} = \begin{bmatrix} \hat{R}_{xx}(0) \\ \vdots \\ \hat{R}_{xx}(q) \end{bmatrix}$$

Modèles MA

- ▶ **2.** Méthode de l'AR long ; on commence par modéliser le signal par un modèle AR très grand ($P > q$) ; on obtient les

paramètres β_j ; on les corrèle : $\pi_I = \sum_{n=0}^{P-|I|} \beta_n \beta_{n+|I|}$; et on résoud un système de ce type :

$$\begin{bmatrix} \pi_0 & \dots & \pi_{q-1} \\ \pi_1 & \ddots & \pi_q \\ \vdots & \ddots & \vdots \\ \pi_{q-1} & \dots & \pi_0 \end{bmatrix} \begin{bmatrix} b_1 \\ \vdots \\ b_q \end{bmatrix} = - \begin{bmatrix} \pi_1 \\ \vdots \\ \pi_q \end{bmatrix}$$

- ▶ **3.** Méthode du maximum de vraisemblance : pas très utilisée, à cause du grand nombre d'optimaux locaux

Modèles ARMA

- ▶ il faut estimer les paramètres a_k et b_l
- ▶ on va faire le lien entre :
 - ▶ la position des zéros et des pôles de $H(z)$
 - ▶ les coefficients ARMA : $\{a_0, \dots\}$ et $\{b_0, \dots\}$
 - ▶ et la DSP obtenue
- ▶ on va jouer avec *aveclesmains_arma.m*

Les modèles ARMA avec les mains

- ▶ essai 1 : on a 6 pôles :
 - ▶ $0.99 \exp(\pm j2\pi 0.05)$ $0.98 \exp(\pm j2\pi 0.20)$ $0.8 \exp(\pm j2\pi 0.3)$
- et on a 2 zéros : de phase égale $\pm 50.4^\circ$; ça correspond donc à un creux de fréquence réduite $\nu = 50.4/360 = 0.14$ ou de fréquence $f = 4480 \text{ Hz}$ si $f_e = 32000 \text{ Hz}$; et d'amplitude 0.95
 - ▶ les pôles sont complexes conjugués 2 à 2 et les zéros sont complexes conjugués \Rightarrow donc le signal est réel
- ▶ voir le cours de traitement de la parole : ça permet de modéliser les formants (pôles) et les anti-formants (zéros) des voyelles nasales

Modèles ARMA

- ▶ l'estimation des paramètres peut se faire de différentes façons ; on ne va pas entrer dans les détails
- ▶ **1.** Estimation des paramètres AR (en réécrivant les équations de Yule-Walker) ; puis MAisation du signal par filtrage ; puis estimation de la partie MA.
- ▶ **2.** Méthode de l'AR long, d'ordre $R = p + q$. La partie MA et la partie AR se séparent après coup. Notez que le problème de ces méthodes << AR long >> vient de ce que plus l'ordre est grand, plus les derniers coefficients sont mal estimés.
- ▶ **3.** Méthode itérative de Durbin. Elle passe par un AR long, et par la méthode de Durbin mise en place pour les MA ci-dessus.
- ▶ **4.** Extension moindre carrés de la méthode 2., en prenant $R \geq p + q$.

Conclusion

- ▶ on a montré que les méthodes dites << à haute résolution >> sont en effet et efficacement des méthodes à haute résolution
- ▶ on a montré qu'on pouvait aisément les utiliser ou bien pour extraire des spectres de raies ou bien pour extraire des enveloppes spectrales (DSP)
- ▶ le parti pris expérimental plutôt que théorique du cours avait pour but de vous montrer de façon concrète les difficultés rencontrées lors de la qualification de méthodes d'analyse (entre autres) : seuls des tests statistiques, en partant ici (pour l'analyse spectrale) de signaux parfaitement calibrés, peut nous donner une idée des performances de tel ou tel outil

N'oubliez pas l'examen

Oral le 29 et le 30 octobre : n'oubliez pas d'aller vous inscrire chez Véronique Baudot, pour votre horaire de passage (vous passez en groupe) ; 30 mn par groupe environ

il y aura quelques questions plus théoriques

Les slides sont disponibles ici

[http://www.metz.supelec.fr/metz/personnel/
rossignol/slides_analyse.pdf](http://www.metz.supelec.fr/metz/personnel/rossignol/slides_analyse.pdf)

Version définitive mise en ligne demain matin

fin