ENS Lyon
Master de Physique - L3
2012–2013

Traitement du Signal

– Analyse spectrale pratique –

Introduction

L'analyse spectrale d'un signal consiste à calculer la quantité d'énergie (ou de puissance) contenue dans les différentes composantes fréquentielles du signal. Pour cela, on associe au signal $x(t)$ une fonction de la fréquence appelée densité spectrale d'énergie (DSE) ou de puissance (DSP) suivant la nature de $x(t)$. Cette grandeur est notée $\Gamma_x(\nu)$. $\Gamma_x(\nu)d\nu$ mesure la quantité d'énergie (respectivement de puissance) contenue par $x(t)$ dans la bande de fréquence $[\nu, \nu + d\nu[$. L'analyse spectrale recouvre l'ensemble des techniques d'estimation de cette densité spectrale. On pourra par exemple définir la puissance spectrale directement à partir de la transformée de Fourier du signal :

- Pour un signal à énergie finie
  On a $E_x = \int_{-\infty}^{+\infty} |x(t)|^2 dt = \int_{-\infty}^{+\infty} |X(\nu)|^2 d\nu$ (théorème de Parseval) où $X(\nu)$ est la transformée de Fourier de $x(t)$. La densité spectrale d'énergie est donc $\Gamma_x(\nu) = |X(\nu)|^2$.

- Pour un signal périodique de période $T = 1/f_0$
  On a $x(t) = \sum_{-\infty}^{+\infty} X_k e^{i2\pi \frac{t}{T}}$ où X_k sont les coefficients de Fourier de $x(t)$. Le spectre est donc un spectre de raies avec de l'énergie seulement aux fréquences $\nu = kf_0$. La densité spectrale est $\Gamma_x(\nu) = |X_k|^2 \delta(\nu - kf_0)$ quelque soit $\nu \in \mathbb{R}$.

- Pour un signal à puissance finie
  On a $P_x = \lim_{T\to+\infty} \frac{1}{T} \int_{-T/2}^{T/2} |x(t)|^2 dt$. Si on définit par $x_T(t)$ la restriction de $x(t)$ à l'intervalle $[-\frac{T}{2}, \frac{T}{2}]$ et $X_T(\nu)$ sa transformée de Fourier alors $P_x = \lim_{T\to+\infty} \frac{1}{T} \int |X_T(\nu)|^2 d\nu$ et $\Gamma_x(\nu) = \lim_{T\to+\infty} \frac{1}{T} |X_T(\nu)|^2$.

Pour les signaux stationnaires, il est aussi possible d'utiliser le théorème de Wiener-Kinchine qui nous dit que la densité spectrale $\Gamma_x(\nu)$ est égale à la transformée de Fourier de la fonction d'auto-covariance (ou de corrélation) du signal :

$$\Gamma_x(\nu) = \int_{-\infty}^{+\infty} \gamma_x(\tau) e^{-i2\pi\nu\tau} d\tau.$$

La fonction d'auto-covariance $\gamma_x(\tau) = \int_{-\infty}^{+\infty} x(t)x^*(t-\tau)dt$, x^* dénote le conjugué complexe de x.

Les signaux aléatoires stationnaires rentrent dans la catégorie des signaux à puissance finie. La définition de la densité spectrale se fait alors grâce au théorème de Wiener-Kinchine en utilisant une moyenne d'ensemble :

$$\gamma_x(\tau) = \mathbb{E}\{x(t)x^*(t-\tau)\},$$

$$\Gamma(\nu) = \int_{-\infty}^{+\infty} \gamma_x(\tau) e^{-i2\pi\nu\tau} d\tau.$$

Le signe $\mathbb{E}$ correspond à l'espérance mathématique d'une variable aléatoire c'est à dire à la moyenne d'ensemble. Par exemple, pour une variable aléatoire réelle x de fonction de densité de probabilité $p(x)$, on a $\mathbb{E}\{x\} = \int_{\mathbb{R}} xp(x)dx$, $\mathbb{E}\{x^2\} = \int_{\mathbb{R}} x^2 p(x)dx$ ....

Les logiciels Matlab, Scilab et Python

Des logiciels spécifiques ont été développés pour faciliter les calculs numériques : ils utilisent un langage interprété optimisé pour les calculs matriciels intensifs et des outils graphiques de simulation. Ces logiciels contiennent des centaines de fonctions mathématiques avec la possibilité de rajouter interactivement des programmes écrits dans divers langages (FORTRAN, C, C++, ...). Ils possèdent des structures de données sophistiquées incluant les listes, les polynômes, les fractions rationnelles, les systèmes linéaires), un interpréteur et un langage de programmation de haut niveau. On peut citer en particulier :

- Le logiciel propriétaire Matlab (http://www.mathworks.com/) pour lequel il existe de nombreuses boîtes à outils puissantes mais souvent onéreuses. L'outil Simulink offre un environnement graphique pour simuler des systèmes dynamiques ou embarqués.

- Le logiciel libre Octave (http://www.gnu.org/software/octave/) utilise un langage matriciel très proche de celui de Matlab. Il permet de réaliser la pluparts des calculs numériques.

- Le logiciel libre Scilab (http://www.scilab.org/), proche de Matlab pour lequel il existe de nombreuses boîtes à outils libres.

- le logiciel (langage) libre Python extrêmement puissant et pour lequel il existe un très grand nombre de librairies. Des interfaces similaires a matlab existent (telles que Spyder) et sont très bien faites.

Dans ce TP nous utiliserons Scilab, Matlab ou Python (avec l'interface Spyder). Scilab et Python sont moins bien en ce qui concernent le graphisme mais ils sont aussi puissants que Matlab et gratuit. Si, au cours de vos stages, vos collaborateurs utilisent Matlab, le passage a Matlab se fera très facilement.

Avant de commencer, quelques remarques importantes :

- Pour Matlab et Scilab, toutes les variables numériques sont des matrices (un scalaire est une matrice de taille 1x1). Toutes les opérations usuelles: $*, /, \wedge$ sont des opérations matricielles. La version scalaire se décline en rajoutant un point devant : $.*, ./, .\wedge$.
  Exemples : [2 3]*[4 5] donne une erreur, [2 3].*[4 5] donne [8 15] et [2 3]*[4 ; 5] donne 23.

- Il est en général plus intéressant d'écrire vos commandes dans un fichier au lieu de le faire dans la fenêtre principale. En effet, on peut utiliser ces commandes plusieurs fois, et on peut commenter allègrement. Ouvrir un fichier avec la barre d'outils du logiciels qui sauvera un fichier texte d'extension ".m" pour Matlab, ".sce" (ou ".sci") pour Scilab et ".py" pour Python. Pour écrire une ligne de commentaire on met le signe pourcent (%) dans Matlab, le double slash (//) pour Scilab et le diese (#) pour Python.

- L'aide en ligne de ces logiciels est bien faite. Taper *helpdesk* dans matlab et *help* dans Scilab pour afficher le navigateur d'aide. Taper help suivi du nom de commande pour avoir les informations spécifiques à cette commande. Pour Python il existe l'inspecteur d'objet de Spyder très utile ou alors taper *help(nom-de-fonction)*.

Importation de données :

A partir de Synchronie ou de Labview, on sauve généralement nos données dans un fichier au format texte. Si on a un ordinateur avec Windows, faire attention à l'opérateur décimal utilisé (. ou ,). Matlab, Scilab et Python n'utilisent que le point. Faire attention à ne sauver que des valeurs numériques, sans entête ni commentaire. Les paramètres importants de l'acquisition apparaissent généralement dans le nom du fichier . Vérifier le répertoire de travail de votre logiciel.

Dans Matlab :

```
% le repertoire de travail courant
pwd
% changer de répertoire
cd /Users/toto/TDacquisition/
%lister les fichiers présents
ls
% lire les données
x=load('nomdefichier.txt'); % importe les données du fichier nomdedonnée.txt
whos x % donne les informations sur la variable x
temps=x(:,1); A=x(:,2); % assignation.
plot(temps,A) % affichage
```

Dans Scilab :

```
// le repertoire de travail courant
pwd
// changer de répertoire
cd /Users/toto/TDacquisition/
//lister les fichiers présents
ls
// lire les données
x = fscanfMat('nomdefichier.txt');
whos -name x
temps=x(:,1); A=x(:,2);plot(temps,A)
```

Dans Python :

```
import os
# le repertoire de travail courant
os.getcwd()
# changer de répertoire
os.chdir('/Users/toto/TDacquisition/')
#lister les fichiers présents
os.listdir('.')

import numpy as np
import matplotlib.pyplot as plt

# lire les données
x = np.loadtxt("nomdefichier.txt")
temps=x[:,1]; A=x(:,2); # assignation.
fig1 =plt.figure()
plt.plot(m.transpose())
# ou
plt.plot(temps,x,'r.-')
plt.ylabel('mes valeurs')
plt.xlabel("temps")
plt.grid()
plt.show()
```

La transformée de Fourier

La définition de la transformée de Fourier est pour toutes fonctions $x \in L^2(\mathbb{R})$:

$$X(\nu) = \int_{-\infty}^{+\infty} x(t) e^{-i2\pi\nu t} dt.$$

Pratiquement, nous n'avons pas le signal au complet mais un ensemble fini de points $x_i = x(t_i)$ avec généralement $t_i = \frac{i}{f_e}, i = 1, ..., N$ (où f_e est la fréquence d'échantillonnage et N le nombre total de points). De la même manière nous ne pouvons calculer la transformée de Fourier que pour un nombre fini de fréquences. Nous calculons donc les coefficients des séries de Fourier suivant :

$$X_k = \sum_{j=0}^{N-1} x_j e^{-i2\pi k \frac{j}{N}}, \text{ pour } k = 0, ..., N-1. \tag{1}$$

On a la formule d'inversion suivante :

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{i2\pi \frac{kn}{N}}, \text{ pour } n = 0, ..., N-1. \tag{2}$$

Le calcul numérique des coefficients de Fourier X_k s'obtiennent par l'algorithme de FFT (pour Fast Fourier Transform) dont la complexité est en $N \log N$ alors que l'utilisation directe de la formule (1) a une complexité de N^2.

x est notre signal de N points (x matrice de taille (N,1)).

Dans Matlab :
$N = 1024$;
$x = randn(1, N)$;% *synthèse d'un bruit blanc gaussien.*
y=fft(x); % est un vecteur de N points contenant les coefficients de Fourier X_k, $k = 0, ..., N-1$.
xr=ifft(y); % est alors équivalent au vecteur x.
plot(x,'r.');hold on;plot(xr,'k');
Dans Scilab :
$N = 1024; x = rand(1, N, 'normal'); y = fft(x); xr = fft(y, 1);$
plot(x,'r.'); plot(xr,'k')

Dans Python :
$N = 1024; x = np.random.randn(N); y = np.fft.fft(x);$
xr=np.fft.ifft(y)
plt.plot(x,'r.'); plt.plot(xr,'k')
plt.show()

Il nous faut maintenant relier le coefficient X_k à la fréquence correspondante. Pour cela il suffit de remarquer que l'équation 1 peut s'écrire

$$X_k = \sum_{j=0}^{N-1} x_j e^{-i2\pi (k \frac{f_e}{N}) \frac{j}{f_e}}, \text{ pour } k = 0, ..., N-1.$$

Cette équation définit les coefficients de Fourier du signal $x(t)$ aux fréquences $\nu = k\frac{f_e}{N}$. Il faut remarquer que les coefficients de Fourier sont périodiques de période f_e et les fréquences $\nu > f_e/2$ correspondent aussi aux fréquences négatives $\nu = k\frac{f_e}{N} - f_e$.

Dans Matlab ou Scilab :
*freq=[0:N/2-1 -N/2:-1]*fe/N; % définition du vecteur fréquence si N pair*
*freq=[0:(N-1)/2 -(N-1)/2:-1]*fe/N; % si N impair*
sp=(abs(fft(x))).^2; % estimation du spectre
plot(freq,sp); % affichage

Dans Python :
sp=np.power(np.abs(np.fft.fft(x)),2)
freq=np.fft.fftfreq(N,1/fe)
plt.plot(freq,sp)
plt.show()

Pour un signal aléatoire stationnaire il faut estimer la densité spectrale en utilisant une moyenne d'ensemble : $\Gamma(\nu) = \mathbb{E}\{|X(\nu)|^2\}$. En supposant le signal ergodique, on estime cette moyenne en découpant le signal de départ en M parties et ensuite on moyenne les puissances spectrales obtenues sur chaque partie du signal : $\Gamma(\nu) = \frac{i}{M}\sum_{i=1}^{M}|X_i(\nu)|^2$ où $X_i(\nu)$ désignent la transformée de Fourier de la i-éme partie. Cette méthode d'estimation spectrale s'appelle la méthode du périodogramme.

Périodogramme : pour un signal aléatoire, stationnaire et ergodique x (x matrice de dimension (N,1)), il faut découper le signal en M parties, estimer le spectre pour chaque partie et moyenner les spectres obtenus.

Dans Matlab :
$N = 1024$*; M* $= 8$;
x=randn(1,N);
xb=reshape(x,N/M,M); % xb est une matrice de dimension (N/M,M).
s=mean(abs(fft(xb)).^2,2);% pour obtenir la moyenne des M spectres.

Dans Scilab :
$N = 1024$*; M* $= 8$;
x=rand(1,N,'normal');
xb=matrix(x,N/M,M);
s=mean(abs(fft(xb,-1,N/M,1)).^2,2);

Dans Python :
N=1024; M=8;
x=np.random.randn(N);
xb=x.reshape(M,N/M);
s=np.power(abs(np.fft.fft(xb)),2);
s=np.mean(s,0);

Si on veut multiplier chaque partie par une fenêtre, on utilise la fonction d'estimation par periodogramme 'pwelch' de Matlab, 'pspect' de Scilab ou 'matplotlib.psd' dans Python.

Covariance et corrélation

Signaux déterministes

La fonction d'autocorrélation d'un signal stationnaire $x(t)$ est définie par :

$$\gamma_x(\tau) = \int_{-\infty}^{+\infty} x(t)x^*(t-\tau)dt.$$

Différentes méthodes d'estimation de $\gamma_x(\tau)$ sont envisageables :

1. La fonction d'autocorrélation biaisée :

$$C_x(\tau) = \frac{1}{N} \sum_{k=1}^{N-\tau} x(k+\tau)x(k). \tag{3}$$

2. La fonction d'autocorrélation non biaisée :

$$C_x(\tau) = \frac{1}{N-\tau} \sum_{k=1}^{N-\tau} x(k+\tau)x(k). \tag{4}$$

où τ et un entier compris entre 0 et $N-1$.

x est notre signal (x matrice de taille (N,1)). On choisi Ntau<N.

Dans Matlab :
N=1024; Ntau=100;
x=randn(1,N);
[c, tau] = xcorr(x, Ntau); % estime la fonction de corrélation.
plot(tau,c); % pour afficher.
Une option de la fonction xcorr permet de choisir l'estimateur biaisé ou non biaisé.

Dans Scilab;
N=1024; Ntau=100; x=rand(1,N,'normal');
c=corr(x,Ntau);tau=0:Ntau-1; plot(tau,c);
L'estimateur non biaisé est utilisé.

Dans Python;
N=100;
x=np.random.randn(N);
y=np.correlate(x,x,mode='full');
c=y/N;
 # nous donne l'estimateur biaisé
tau=np.arange(-N,N-1,dtype=float);
plt.plot(tau,c)
plt.grid()
plt.show()

Signaux aléatoires

Soit x une variable aléatoire, le moment d'ordre p de la variable aléatoire x est défini à partir d'une moyenne d'ensemble et se note $\mathbb{E}\{x^p\}$. Si $p(x)$ est la fonction densité de probabilité de x alors $\mathbb{E}\{x^p\} = \int x^p p(x)dx$. La variance de x est défini par $Var_x = \mathbb{E}\{x^2\} - \mathbb{E}\{x\}$ et l'écart type par $\sigma_x = \sqrt{Var_x}$.

Si on dispose de N valeurs d'une variable aléatoire discrète x, $\{x_i\}_{i=1..N}$, on estime $\mathbb{E}\{x\}$ par $m_x = \frac{1}{N}\sum_{i=1}^{N} x_i$, $\mathbb{E}\{x^2\} = \frac{1}{N}\sum_{i=1}^{N} x_i^2$ ... La fonction de densité de probabilité $p(x)$ peut être estimé par la méthode des histogrammes. Cette méthode consiste à compter le nombre de fois N_x que la variable x a une valeur dans l'intervalle $[x, x+\Delta x[$. La densité de probabilité p(x) est alors estimée par $N_x/N/\Delta x$ (normalisation pour avoir $\int p(x)dx = 1$).

On dispose de N échantillons d'un processus x (x est une matrice de taille (N,1)).

Dans Matlab :
```
N=1024; Nbin=100;
x=randn(1,N);
m=mean(x); % l'estimation de E{x};
v=var(x); sigma=std(x); % estimation de la variance et de l'écart type.
[pxbin, xbin] = hist(x, Nbin);% calcul de l'histogramme.
pxbin = pxbin/sum(pxbin)/mean(diff(xbin)); % densité de probabilité
bar(xbin,pxbin); % visualisation
```

Dans Scilab :
```
N=1024; Nbin=100;
x=rand(1,N,'normal');
m=mean(x); v=variance(x); sigma=stdev(x); // moyenne, variance et écart type de x.
Dx=(max(x)-min(x))/(Nbin+1); xbin=min(x)+[0:Nbin]*Dx;
[ind, pxbin] = dsearch(x, xbin);
pxbin = pxbin/sum(pxbin)/Dx;
bar(xbin(1:$-1),pxbin);
```

Dans Python :
```
N=1024;nbin=100;
x=np.random.randn(N);
m=np.mean(x);v=np.var(x); s=np.std(x);
n, bins, patches = plt.hist(x, nbin, normed=1, facecolor='g', alpha=0.75)
plt.xlabel('Données')
plt.ylabel('Probabilite')
plt.title('Histogramme')
#plt.text(60, .025, r'μ = 100, σ = 15')
#plt.axis([40, 160, 0, 0.03])
plt.grid(True)
plt.show()
```

Soit x et y deux variables aléatoires, on défini la covariance par :

$$Cov(x, y) = \mathbb{E}\{(x - \mathbb{E}\{x\})(y - \mathbb{E}\{y\})\}$$

Auto-covariance d'un processus aléatoire

Un processus aléatoire est une variable aléatoire qui dépend du temps, c'est à dire que $x_t = x(t)$ est une variable aléatoire pour tous les instants t. On peut alors définir l'auto-covariance de x par :

$$C_x(t, t') = Cov(x_t, x_{t'}) = \mathbb{E}\{(x_t - \mathbb{E}\{x_t\})(x_{t'} - \mathbb{E}\{x_{t'}\})\} = \mathbb{E}\{x_t x_{t'}\} - \mathbb{E}\{x_t\}\mathbb{E}\{x_{t'}\}.$$

Si notre variable aléatoire x_t est discrète, $t = 1 : N$, l'auto-covariance est une matrice C dont les éléments sont $C_{ij} = \mathbb{E}\{x_i x_j\} - \mu_i \mu_j, 1 \leq i \leq N, 1 \leq j \leq N$. Pour estimer l'espérance il faut pour tous les instants t plusieurs réalisations de x_t. Si on note x_t^k la k-ième réalisation ($k = 1..K$) de x au temps t, on estime $C_{i,j}$ par

$$C_{i,j} = \frac{1}{K} \sum_{k=1}^{K} x_i^k x_j^k - \left(\frac{1}{K} \sum_{k=1}^{N} x_i^k \right) \left(\frac{1}{K} \sum_{k=1}^{N} x_j^k \right). \tag{5}$$

On notera que la moyenne de notre processus x intervient de manière non triviale dans ces équations. De plus, le biais de l'estimation de la covariance (et de la corrélation) dépend de cette moyenne. Il est donc fortement conseillé d'estimer la covariance ou la corrélation sur des variables centrées c'est à dire de moyenne nulle.

On dispose de Nreal réalisations de taille N du processus x (x est une matrice de taille (Nreal,N)).

Dans Matlab :
Nreal=100;N=1024;
x=randn(Nreal,N);
c=cov(x); % matrice de covariance.
imagesc(c); % visualisation de cette matrice.

Dans Scilab :
Nreal=100;N=1024;
x=rand(Nreal,N,'normal');
c=mvvacov(x); // matrice de covariance. Pour visualiser cette matrice, il faut adapter les valeurs de la matrice à la palette de couleur (colormap). On décide de coder la couleur avec 128 pixels.
xset('colormap',hotcolormap(128));
*Matplot((c-min(c))/(max(c)-min(c))*128); // visualisation de la matrice en adaptant ses valeurs .*

Dans Python :
Nreal=100;N=1024;
x=np.random.randn(N,Nreal);
c=np.cov(x)
uu=plt.imshow(c)
plt.colorbar(uu)
plt.show()

Si notre signal est supposé stationnaire, les différentes statistiques ne dépendent pas du temps ($\mathbb{E}\{x_t\} = \mathbb{E}\{x\}$, $\mathbb{E}\{x_t^2\} = \sigma^2$ ...) et l'auto-covariance ne dépend alors que de la différence des deux temps $\tau = t - t'$:

$$C_x(t, t') = C_x(t - t') = \mathbb{E}\{x_t x_{t'}\} - \mathbb{E}\{x\}^2;$$

Si le signal est supposé ergodique, la moyenne d'ensemble peut être remplacée par une moyenne temporelle : les $C_x(t - t')$ peuvent être obtenus avec une seule réalisation de x en utilisant les équations (3) ou (4).

Si on a plusieurs réalisations du processus, on réalise une moyenne temporelle et une moyenne sur les réalisations. On remarque que dans ce cas la matrice de covariance $C_{i,j}$ est symmétrique et vérifie la relation $C_{i,i+\tau} = C_{j,j+\tau}$. Toutes les lignes de cette matrice sont donc des estimations de la fonction d'autocovariance et en moyennant suivant les diagonales on obtient notre estimé final de l'auto-covariance.

0n dispose de Nreal réalisations de taille N du processus x (x est une matrice de taille (Nreal,N)).

Dans Matlab :

```
Nreal=100;N=1024; Ntau=100;
x=randn(Nreal,N);
for i=1:Nreal; % boucle sur les réalisations
    cxx=xcorr(x(i,:),Ntau);% estimation de la fonction de corrélation de la réalisation i.
    ctot(i,:)=cxx;
end
c=mean(ctot);% moyenne des fonctions de corrélation obtenus.
plot(tau,c);
```

Dans Scilab :

```
Nreal=100;N=1024; Ntau=100;
x=rand(Nreal,N,'normal');
for i=1:Nreal
    cxx=corr(x(i,:),Ntau);
    ctot(i,:)=cxx;
end
c=mean(ctot,'r');
tau=0:Ntau-1; plot(tau,c);
```

Dans Python :

```
Nreal=100;N=1024;
x=np.random.randn(Nreal,N);
tau=np.arange(-N,N-1,dtype=float);
ctot=np.zeros((Nreal,2*N-1) ,dtype=float);
for i in range(Nreal) :
    # pas tres rapide. Calcul en Fourier plus rapide
    cxx=np.correlate(x[i,:],x[i,:],mode='full');
    ctot[i,:]=cxx/N;

c=np.mean(ctot,0)
plt.plot(tau,c)
plt.grid()
plt.show()
```

En passant par la matrice de covariance :

Dans Matlab :

```
Nreal=100;N=1024;
x=randn(Nreal,N);
c=cov(x);
for i=-N+1:N-1
    cor(i+N)=mean(diag(c,i));% on prend la moyenne des diagonales de la matrice
end;
tau=-N+1:N-1;plot(tau,cor)
```

Dans Scilab :

```
Nreal=100;N=1024;
x=rand(Nreal,N,'normal');
c=mvvacov(x);
for i=-N+1:N-1,
    cor(i+N)=mean(diag(c,i));
end;
tau=-N+1:N-1;plot(tau,cor)
```

Dans Python :

```
import scipy as sc
Nreal=100;N=1024;
x=np.random.randn(N,Nreal);
c=np.cov(x)
tau=np.arange(-(N-1),N,dtype=float);
cor=np.zeros(2*N-1 ,dtype=float);

for i in range(-N+1,N-1) :
    cor[i+N]=np.mean(sc.diag(c,i));

tau=np.arange(-N,N-1,dtype=float);
plt.plot(tau,cor)
plt.grid()
plt.show()
```

On obtient finalement la densité spectrale en calculant la transformée de Fourier de la fonction de corrélation obtenu.

Covariance de deux processus aléatoires

On défini la covariance de deux processus aléatoires x et y par :

$$C_{xy}(t,t') = Cov(x_t, y_{t'}) = \mathbb{E}\{(x_t - \mathbb{E}\{x_t\})(y_{t'} - \mathbb{E}\{y_{t'}\})\} = \mathbb{E}\{x_t y_{t'}\} - \mathbb{E}\{x_t\}\mathbb{E}\{y_{t'}\}.$$

Si les deux processus sont supposés stationnaires la covariance ne dépend que de la différence des deux temps :

$$C_{xy}(t,t') = C_{xy}(t-t') = \mathbb{E}\{x_t y_{t'}\} - \mathbb{E}\{x\}\mathbb{E}\{y\}.$$

Si on suppose l'ergodicité des signaux x et y, la moyenne d'ensemble peut être remplacée par une moyenne temporelle. Dans ce cas, le coefficient de corrélation des processus x et y se défini par :

$$r_{xy} = \frac{C_{xy}(0)}{\sigma_x \sigma_y},$$

où $\sigma_x = \mathbb{E}\{(x - \mathbb{E}\{x\})^2\}$ et $\sigma_y = \mathbb{E}\{(y - \mathbb{E}\{y\})^2\}$ sont respectivement l'écart-type du processus x et y.
Si $y = x$ on a $r_{xy} = 1$ et si $y = -x$ $r_{xy} = -1$

0n dispose de Nreal réalisations de taille N du processus x et y (x et y sont des matrices de taille (Nreal,N)).

Dans Matlab :
```
Nreal=100;N=1025;
x=randn(Nreal,N);y=x(:,2:N);x=x(:,1:N-1);%y c'est x retarde de 1/ N=1024;
c=cov([x y]);% la matrice sera égal a [cxx cxy; cyx cxx]
cxy=c(N+1:end,1:N);% on récupère le quart supérieur droit : la matrice de covariance de x et y
imagesc(c); % le maximum n'est pas sur la diagonale mais décalé de 1.
```

Dans Scilab :
```
Nreal=100;N=1025;
x=rand(Nreal,N,'normal');y=x(:,2:N);x=x(:,1:N-1);
N=1024;
stacksize(8900000);% pour augmenter la mémoire disponible.
c=mvvacov([x y]);
cxy=c(N+1:$,1:N);
xset('colormap',hotcolormap(128));
Matplot((cxy-min(cxy))/(max(cxy)-min(cxy))*128);
```

Pour obtenir la fonction de covariance on peut faire la moyenne suivant les diagonales de la matrice de covariance obtenue précédemment ou faire appel aux fonctions xcorr pour Matlab et corr pour Scilab sur chaque réalisation et faire la moyenne ensuite.

Dans Python :
```
Nreal=100;N=1025;
x=np.random.randn(N,Nreal);
y=x[2:N,:];x=x[1:N-1,:]; #y c'est x retarde de 1
cxy=np.cov(x,y)

fig1 =plt.figure()
uu=plt.imshow(cxy[0:1023,1:1023])
plt.colorbar(uu)
plt.show()
fig2 =plt.figure()
uu=plt.imshow(cxy[1024:2046,1024:2046])
plt.colorbar(uu)
plt.show()
fig3 =plt.figure()
uu=plt.imshow(cxy)
plt.colorbar(uu)
plt.show()
```