A Stroll with Random Signals and IPython

Jean-François Bercher

ESIEE-Paris

Contents

1 Random Signals

1.1 Introduction to Random Signals

Auteur: J.-F. Bercher date: october, 2014 Last update: october, 2015

Just as a random variable is a set of values associated with a probability distribution, a random signal, also callled a random process, is a set of *functions* associated with a probability distribution. In addition to properties similar to those of random variables, the study of random processes include characterizations of dependences *in* the random process (namely notions of *correlation*), the study the behaviour of the function under transformations (*filtering*) and the design of some optimal transformations.

Notations

We denote by $X(n,\omega)$ a random signal X. It is a set of functions of n, the set being indexed by ω. A random signal is thus a bivariate quantity. When $\omega = \omega_i$ is fixed, we get a *realization* of the random process, denoted $X(n,\omega_i)$ or, more simply $x_i(n)$. When n is fixed, the random process reduces to a simple random variable. Considering the process for $n = n_i$, we obtain a random variable $X(n_i,\omega)$, denoted $X_i(\omega)$, or X_i. Finally, we will denote x_i the values taken by the random variable X_i.

1.2 Fundamental properties

1.2.1 Stationnarity

Definition 1. *A random signal is said stationnary if its statistical properties are invariant by time translation.*

This means that the joint probability density function

$$p_{X(n_1),X(n_2),\ldots,X(n_k)} = p_{X(n_1-\tau),X(n_2-\tau),\ldots,X(n_k-\tau)}, \tag{1.1}$$

and if $\tau = n_k$

$$p_{X(n_1),X(n_2),\ldots,X(n_k)} = p_{X(n_1-n_k),X(n_2-n_k),\ldots,X(0)}. \tag{1.2}$$

Therefore, the joint distribution only depends on $k-1$ parameters, instead of the k initial parameters.

As a consequence, we have that

- $\mathbb{E}[X(n)] = \mu$ is constant and does not depend on the particular time n

- $\mathbb{E}[X(n)X(n-\tau)^*] = R_X(\tau)$ only depends on the delay between the two instants. In such a case, the resulting function $R_X(\tau)$ is called a **correlation function**.

1.2.2 Ergodism

Definition

The time average, taken on realization ω is

$$\langle X(n,\omega)^n \rangle = \lim_{N\to+\infty} \frac{1}{N} \sum_{[N]} X(n,\omega)^n. \tag{1.3}$$

Of course, in the general case, this time average **is** a random varaible, since it depends on ω.

Definition A random signal is said ergodic if its time averages are deterministic, i.e. non random, variables.

Important consequence

A really important consequence is that if a signal is both stationnary and ergodic, then the statistical means and the time averages are equal.

$$\boxed{\mathbb{E}[\bullet] = \langle \bullet \rangle} \tag{1.4}$$

Exercise > - Check that

- (moments) Check that if the signal is both stationnary and ergodic, then

$$\mathbb{E}[X(n,\omega)^n] = \lim_{N\to+\infty} \frac{1}{N} \sum_{[N]} X(n,\omega)^n, \tag{1.5}$$

- (covariance) Similarly, check that

$$R_X(\tau) = \mathbb{E}[X(n,\omega)X(t-\tau,\omega)] = \lim_{N\to+\infty} \frac{1}{N} \sum_{[N]} X(n,\omega)X(t-\tau,\omega). \tag{1.6}$$

1.2.3 Examples of random signals

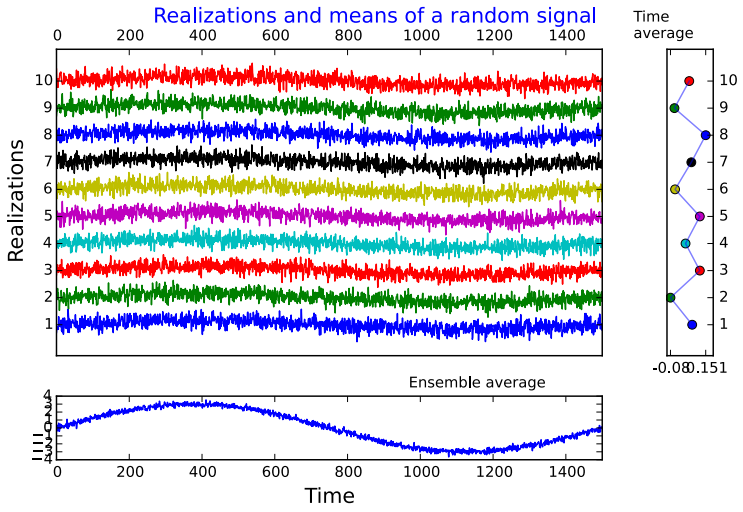
1. Let us first consider the noisy sine wave $X(n,\omega) = A\sin(2\pi f_0 n) + B(n,\omega)$. Function `plot_rea` plots some realizations of this signal and plots the ensemble and time averages. You will also need `sig_histo` which plots the histogram, together with the time series.

   ```
   from plot_rea import *
   from plot_sighisto import *
   ```

   Experiment with the parameters (amplitude, number of samples). Is the signal stationary, ergodic, etc?

```
import scipy.stats as stats

M=10; # number of bins in histograms
N=1500 # Number of samples per realization
K=200 # Total number of realizations

XGauss=stats.norm(loc=0,scale=1)

#Sine wave plus noise
X=3*XGauss.rvs(size=(K,N))+3*np.outer(np.ones((K,1)),np.sin(2*np.pi*np.
    arange(N)/N))
print("Standard deviation of time averages: ",np.std(np.mean(X,axis=1)))
#pylab.rcParams['figure.figsize'] = (10.0, 8.0)
plt.rcParams['figure.figsize'] = (8,5)
plot_rea(X,nb=10,fig=1)
```

```
Standard deviation of time averages:  0.0751828360059
```

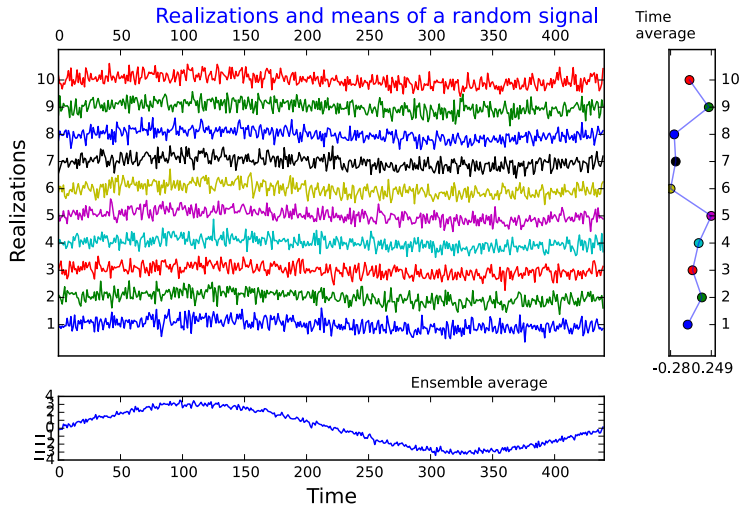


By varying the number of samples N, we see that the time average converges to zero, for each realization. Thus we could say that this process is ergodic. However, the ensemble average converges to the sine wave and is dependent if time: the process *is not stationary*.

```
XGauss=stats.norm(loc=0,scale=1)
#pylab.rcParams['figure.figsize'] = (10.0, 8.0)
plt.rcParams['figure.figsize'] = (8,5)

def q1_experiment(N):
    K=200
    #Sine wave plus noise
    X=3*XGauss.rvs(size=(K,N))+3*np.outer(np.ones((K,1)),np.sin(2*np.pi*np
        .arange(N)/N))
    print("Standard deviation of time averages: ",np.std(np.mean(X,axis=1)
        ))
    plot_rea(X,nb=10,fig=1)
interact(q1_experiment, N=(0,2000,10))
```

```
Standard deviation of time averages:  0.148465583595
```

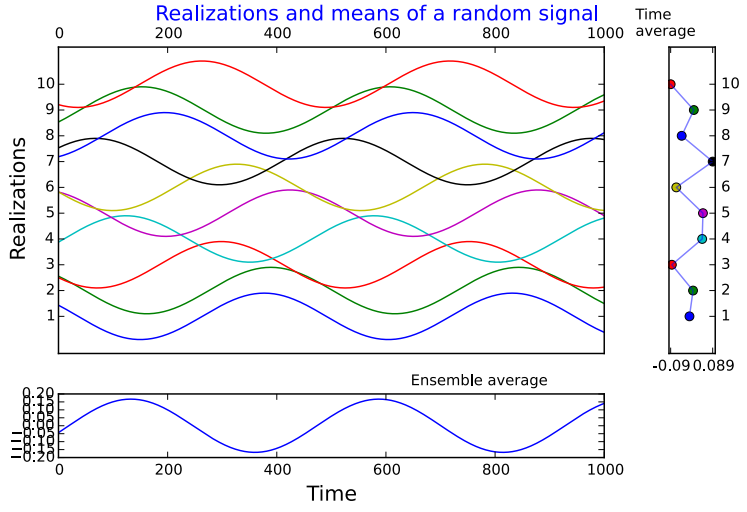
```
None
```



2- Consider now a sine wave with a random phase $X(n,\omega) = A\sin(2\pi f_0 n + \phi(\omega))$.

Experiment with the parameters (amplitude, number of samples). Is the signal stationary, ergodic, etc? Also change the value of the frequency, and replace function `sin` by `square` which generates a pulse train instead of a sine wave.

```
from pylab import *
K=100
N=1000
fo=2.2/N
S=zeros((K,N))
for r in range(K):
    S[r,:]=1.1*sin(2*pi*fo*arange(N) + 2*pi*rand(1,1));
plot_rea(S,fig=2)
```



This example shows that a random signal is not necessarily noisy and irregular. Here we have a random signal which is 'smooth'. The random character is introduced by the random phase, which simply reflects that we do not know the time origin of this sine wave.

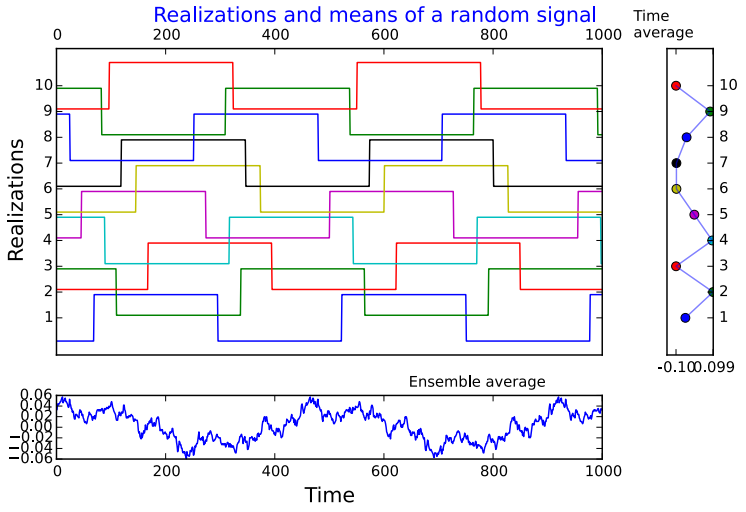
Here, we see that both the time average and the ensemble average converge to zero. Therefore, we can conclude that this signal is stationary and ergodic.

Let us now define a square wave:

```
def square(x):
    """square(x): \n
    Returns a pulse train with period :math:`2\pi`
    """
    return sign(sin(x))
```

Then generate a random square wave as follows

```
K=1000
N=1000
S=zeros((K,N))
for r in range(K):
    S[r,:]=1.1*square(2*pi*fo*arange(N) + 2*pi*rand(1,1));
plot_rea(S,fig=2)
```



Again, we see that both means tend to zero, a constant, which means that the signal is stationary (its ensemble average does not depend of time) and ergodic (its time average does not depend on the actual realization).

1.2.4 White noise

For discrete signals, a white noise is simply a sequence of independent variables (often the variables will be also identically distributed). An independent and identically distributed signal is denoted iid. Since the components of a white noise are all independent, there will be no correlation between them. We will see later that the spectral representation of a white noise is *flat*, thus coining the name of white noise by analogy with the white light.

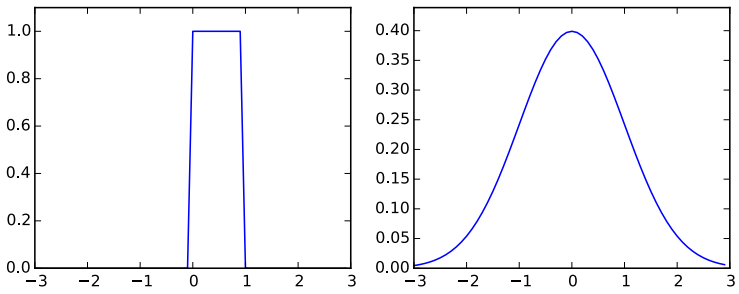
The notion of white noise is more involved in the case of a time-continuous signal. The white noise is in such case a limit processe with "microscopic dependences".

We consider now two kinds of random noises: the first one is a sequence of independent and identically distributed variables (iid variables), according to a uniform distribution. The second one is an iid sequence, Gaussian distributed. Plot the two probability density functions, plot the histograms (with `sig_histo`) and compare the time series.

3- Compute and analyze the histograms of two white noises, respectively with a uniform and a Gaussian probability density function, using the lines in script `q1c`. Do this for several realizations (launch the program again and again) and change the number of points and of bins. Compare the two signals. What do you think of the relation between whiteness and gaussianity.

```
# An object "uniform random variable" with fixed bounds [0,1]
x_uni=stats.uniform(loc=0,scale=1)
# An object "gaussian random variable" with zero mean and scale 1
x_gauss=stats.norm(loc=0,scale=1)
#plt.rcParams['figure.figsize'] = (8,5)
fig, (ax1, ax2) = plt.subplots(1,2,figsize=(8,3))
x=arange(-3,3,0.1)
ax1.plot(x,x_uni.pdf(x)); ax1.set_ylim([0, 1.1*max(x_uni.pdf(x))])
ax2.plot(x,x_gauss.pdf(x)); ax2.set_ylim([0, 1.1*max(x_gauss.pdf(x))])
```

```
(0, 0.43883650844157601)
```



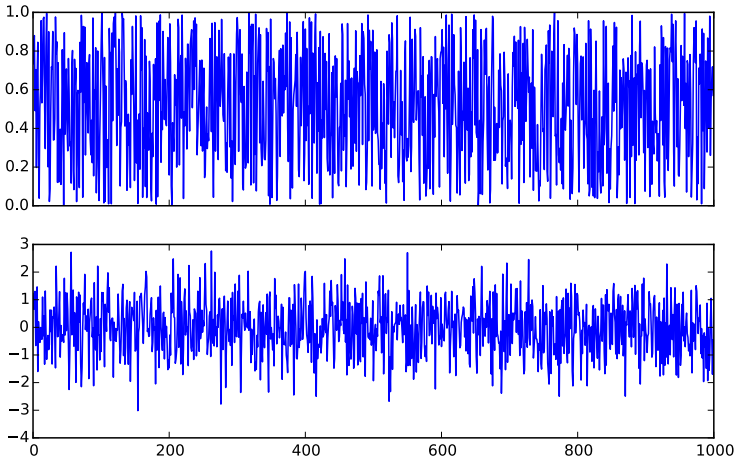

```
(m,v)=x_uni.stats(moments='mv')
print("Uniform distribution: ","Value of the mean : {0:2.3f} and of the
    variance {1:2.3f}".format(float(m),float(v)))
(m,v)=x_gauss.stats(moments='mv')
print("Gauss distribution: ","Value of the mean : {0:2.3f} and of the
    variance {1:2.3f}".format(float(m),float(v)))
```

```
Uniform distribution:  Value of the mean : 0.500 and of the variance 0.083
Gauss distribution:  Value of the mean : 0.000 and of the variance 1.000
```

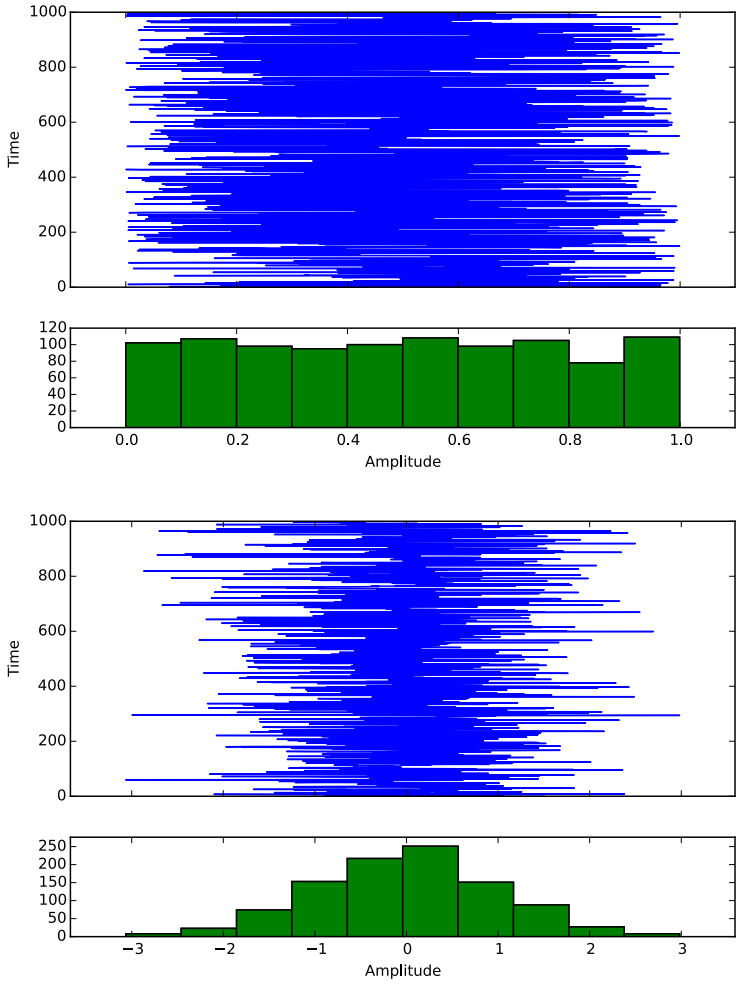
We can compare the two signals

```
fig, (ax1,ax2)=subplots(2,1,sharex=True)
ax1.plot(x_uni.rvs(size=N))
ax2.plot(x_gauss.rvs(size=N))
```

```
[<matplotlib.lines.Line2D at 0x7fe9b0069e48>]
```




```
from plot_sighisto import *
plot_sighisto(x_uni.rvs(size=N),fig=1)
plot_sighisto(x_gauss.rvs(size=N),fig=2)
```



We see that the Gaussian noise is more concentrated on its mean 0, and exhibits more important values, while the uniform noise is confined into the interval [0,1].

Concerning the question on the relation between whiteness and Gaussianity, actually, there is no relation between these two concepts. A white noise can be distributed according to any distribution, and a Gaussian sequence is not necessarily iid (white).

1.3 Second order analysis

1.3.1 Correlation functions

Definition

If $X(n,\omega)$ and $Y(n,\omega)$ sare two jointly stationnary random processes, the intercorrelation and autocorrelation functions are defined by

$$\begin{aligned} R_{XY}(k) &\stackrel{\triangle}{=} \mathbb{E}\left[X(n,\omega)Y^*(n-k,\omega)\right] = \lim_{\text{erg } N\to+\infty} \frac{1}{N}\sum_{n=0}^{N} X(n,\omega)Y^*(n-k,\omega), \\ R_{XX}(k) &\stackrel{\triangle}{=} \mathbb{E}\left[X(n,\omega)X^*(n-k,\omega)\right] = \lim_{\text{erg } N\to+\infty} \frac{1}{N}\sum_{n=0}^{N} X(n,\omega)X^*(n-k,\omega). \end{aligned} \tag{1.7}$$

Main properties

1. (Hermitian symmetry)

$$R_{YX}(\tau) = \mathbb{E}\left[Y(n,\omega)X^*(n-\tau,\omega)\right] = \mathbb{E}\left[Y(n+\tau,\omega)X^*(n,\omega)\right] = \mathbb{E}\left[X(n,\omega)Y^*(n+\tau,\omega)\right]^* = R^*_{XY}(-\tau). \tag{1.8}$$

2. (Symmetry for the autocorrelation). In the case of the autocorrelation function, the hermitian symmetry reduces to

$$R_{XX}(\tau) = R^*_{XX}(-\tau). \tag{1.9}$$

3. (Centering). If $X_c(n,\omega) = X(n,\omega) - m_X$ is the centered signal, then

$$R_{XX}(\tau) = R_{X_cX_c}(\tau) + m_X^2. \tag{1.10}$$

4. (Autocorrelation and power). For a delay $\tau = 0$, we have

$$R_{XX}(0) = \mathbb{E}\left[|X(n,\omega)|^2\right] = \lim_{\text{erg } N\to+\infty} \frac{1}{N}\sum_{[N]} |X(n,\omega)|^2 = P_X. \tag{1.11}$$

   This shows that $R_{XX}(0)$ is nothing but the power of the signal under study. Observe that necessarily $R_{XX}(0) > 0$.

5. (Maximum). Beginning with the *Schwarz inequality*,

$$|< x, y >|^2 \leq < x, x >< y, y >, \tag{1.12}$$

   and using the scalar product $< x_1, x_2 >= \mathbb{E}\left[X_1(n)X_2^*(n)\right]$, we get

- $|R_{YX}(\tau)|^2 \leq R_{XX}(0)R_{YY}(0), \quad \forall \tau,$

- $|R_{XX}(\tau)| \leq R_{XX}(0), \tilde{} \forall \tau, $

1. (Non negativity) The autocorrelation function is non negative definite

$$\sum_i \sum_j \lambda_i R_{XX}(\tau_i - \tau_j)\lambda_j \geq 0, \quad \forall i, j. \tag{1.13}$$

proof: develop $\mathbb{E}\left[|\sum_i \lambda_i X(\tau_i)|^2\right] \geq 0$

2. (Correlation coefficient). By the maximum property, the correlation coefficient

$$\rho_{XY}(\tau) = \frac{R_{YX}(\tau)}{\sqrt{R_{XX}(0)R_{YY}(0)}} \tag{1.14}$$

is such that $\rho_{XY}(\tau) \leq 1$.

3. (Memory). If the correlation coefficient is zero after a certain time t_c then the process is said to have a finite memory and t_c is called the *correlation time*.

Exercises

1. Developing $\mathbb{E}\left[|X + \lambda Y|^2\right]$ into a polynom of λ and observing that this polynom in always nonegative, prove the Scharz inequality.

2. Consider a random signal $U(n, \omega)$ defined on the interval $[0, N]$. Define the periodic signal

$$X(n, \omega) = \mathrm{Rep}_N[U(n, \omega)] = \sum_k U(t - kN, \omega). \tag{1.15}$$

- Show that $R_{UU}(\tau) = 0$ for $\tau \notin [-N, N]$.
- Show that $R_{XX}(\tau)$ is a periodic function with period N and express $R_{XX}(\tau)$ as a function of $R_{UU}(\tau)$.

2. Consider a random signal $X(n, \omega)$ with autocorrelation $R_{XX}(k)$ and define

$$Z(n, \omega) = X(n, \omega) + aX(n - n_0, \omega). \tag{1.16}$$

Compute the autocorrelation function of $Z(n, \omega)$.

Estimation of correlation functions

By stationnarity and ergodism, we have that

$$R_{XX}(k) \underset{\text{erg}}{=} \lim_{N \to +\infty} \frac{1}{N} \sum_{n=0}^{N} X(n, \omega)X^*(n - k, \omega). \tag{1.17}$$

Given a finite number of points N, with data known from $n = 0..N - 1$, it is thus possible to approximate the correlation function by a formula like

$$R_{XX}(k) = \frac{1}{N} \sum_{n=0}^{N-1} X(n, \omega)X^*(n - k, \omega). \tag{1.18}$$

If we take $k \geq 0$, we see that $X^*(n-k,\omega)$ is unavailable for $k > n$. Consquently, the sum must go from $n = k$ to $N-1$. At this point, people define two possible estimators. The first one is said "unbiased" while the second is "biased" (check this by computing the expectation $\mathbb{E}[\bullet]$ of the two estimators).

$$\hat{R}_{XX}^{(unbiased)}(k) = \frac{1}{N-k}\sum_{n=k}^{N-1} X(n,\omega)X^*(n-k,\omega) \tag{1.19}$$

$$\hat{R}_{XX}^{(biased)}(k) = \frac{1}{N}\sum_{n=k}^{N-1} X(n,\omega)X^*(n-k,\omega). \tag{1.20}$$

For the biased estimator, it can be shown (Bartlett) that the variance has the form

$$Var\left[\hat{R}_{XX}^{(biased)}(k)\right] \approx \frac{1}{N}\sum_{m=-\infty}^{+\infty} \rho(m)^2+\rho(m+k)\rho(m-k)-4\rho(m)\rho(k)\rho(m-k)+2\rho(m)^2\rho(k)^2, \tag{1.21}$$

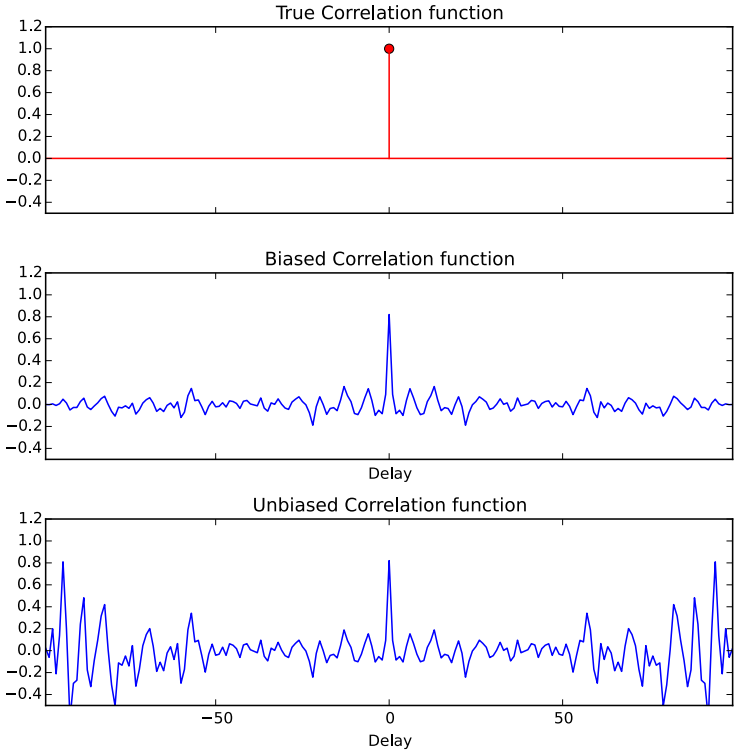
that is, essentially a constant over N. As far the unbiased estimator is concerned, we will have a factor $N/(N-k)$, and we see that this time the variance *increases* with k. Thus, though it is unbiased, this estimator has a very bad behaviour with respect to the variance.

This is checked below. First we generate a gaussian white noise, compute the two estimates of the correlation function and copare them.

```
from correlation import xcorr
from scipy import stats as stats
N=100
XGauss=stats.norm(loc=0,scale=1)
S=XGauss.rvs(size=N)
#
Rbiased,lags=xcorr(S,norm='biased')
Runbiased,lags=xcorr(S,norm='unbiased')
Rtheo=zeros(size(Rbiased))
Rtheo[lags==0]=1
```

```
Rt=ones(1)
fig,ax=subplots(3,1,figsize=(7,7),sharex=True,sharey=True)
# biased correlation
ax[1].plot(lags,Rbiased)
#ax[0].axvline(0,ymin=0,ymax=1,color='r',lw=3)
ax[1].set_title("Biased Correlation function")
ax[1].set_xlabel("Delay")
ax[1].axis('tight')  #Tight layout of the axis
# unbiased correlation
ax[2].plot(lags,Runbiased)
ax[2].set_title("Unbiased Correlation function")
ax[2].set_xlabel("Delay")
# theoretical correlation
ax[0].stem([0],[1],linefmt='r-', markerfmt='ro', basefmt='r-')
ax[0].plot([lags[0],lags[-1]],[0, 0],'r')
ax[0].set_title("True Correlation function")
fig.tight_layout()
ax[1].axis('tight')
ax[0].set_ylim([-0.5,1.2])
```

```
(-0.5, 1.2)
```

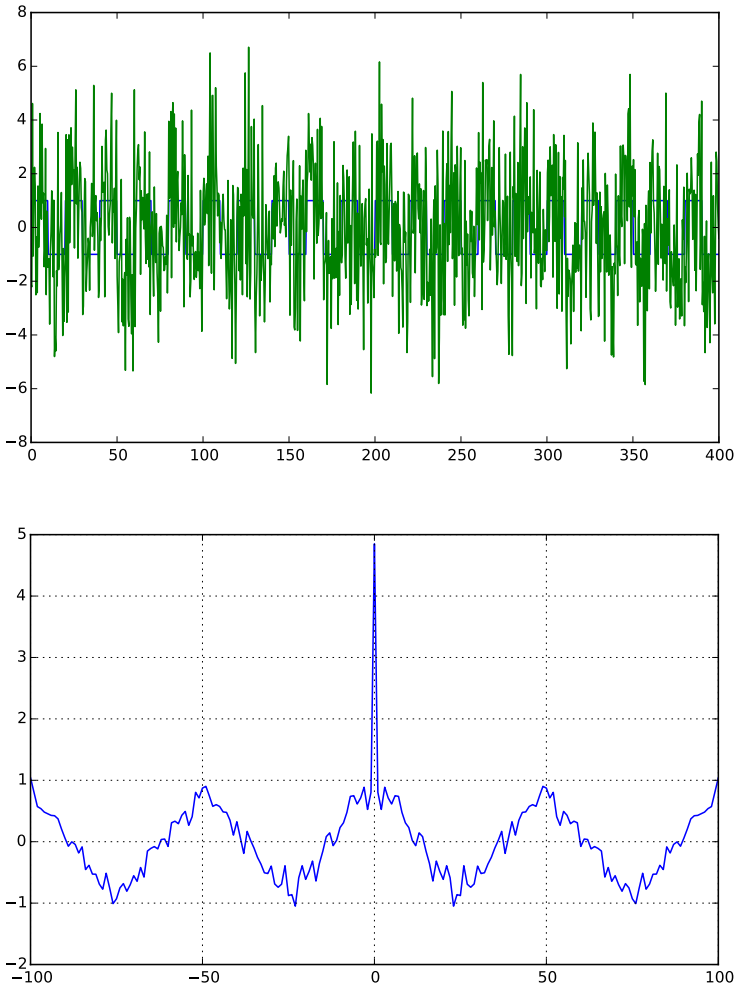


Detecting hidden periodicities

We consider here a time series composed of a periodic signal corrupted by a white noise. The signal is completely hidden by the noise. We show here that it is possible to find some information in the autocorrelation function.

Exercises - (a) Check that the correlation of a periodic signal is periodic - (b) Give the correlation of $y = s + w$ if s and w are independent.

```
def square(x):
    """square(x): \n
    Returns a pulse train with period :math:`2\pi`
    """
    return sign(sin(x))

N=1000
f0=0.05
t=np.linspace(0,400,N)
x=square(2*pi*f0*t)
noise=stats.norm(loc=0,scale=2).rvs(N)
observation=x+noise
#
plt.plot(t,x,'-')
plt.plot(t,observation)
#
Rbiased,lags=xcorr(observation,norm='biased',maxlags=100)
plt.figure()
plt.plot(lags,Rbiased)
plt.grid(b='on')
```



The last figure shows the correlation of the noisy periodic signal. This correlation is simply the superposition of the correlation of the noise and of the correlation of the signal (Check it!)

$$R_{\text{obs,obs}} = R_{\text{sig,sig}} + R_{\text{noise,noise}} \tag{1.22}$$

Since the correlation of the noise (a Dirac impulse) is concentrated at zero, we can read - the period of the signal: 50 (that is a relative frequency of 50/1000=0.05) - the power of the signal: 1 - the power of the noise: 5 - 1 = 4 (was generated with a standard deviation of 2). The correlation function then enable us to grasp many informations that were not apparent in the time series!

1.4 Filtering

1.4.1 General relations for cross-correlations

We consider a situation where we want to study the correlations between the different inputs and outputs of a pair of two filters:

$$\left\{ \begin{array}{rcl} Y_1(n,\omega) & = & (X_1 * h_1)(n,\omega), \\ Y_2(n,\omega) & = & (X_2 * h_2)(n,\omega), \end{array} \right. \tag{1.23}$$

Let us compute the intercorrelation between $Y_1(n)$ and $Y_2(n)$:

$$R_{Y_1Y_2}(m) = \mathbb{E}\left[Y_1(n,\omega)Y_2^*(n-m,\omega))\right] = \mathbb{E}\left[(X_1 * h_1)(n,\omega))(X_2^* * h_2^*)(n-m,\omega))\right]. \tag{1.24}$$

The two convolution products are

$$\begin{aligned}(X_1 * h_1)(n,\omega)) &= \sum_u X_1(u,\omega))h_1(n-u),\\ (X_2 * h_2)(n-m,\omega)) &= \sum_v X_2(v,\omega))h_2(n-m-v),\end{aligned}$$

and

$$\begin{aligned}R_{Y_1Y_2}(m) &= \mathbb{E}\left[\sum_u X_1(n-u,\omega))h_1(u)\sum_v X_2^*(n-m-v,\omega))h_2^*(v)\right]\\ &= \mathbb{E}\left[\sum_u\sum_v X_1(n-u)h_1(u)X_2^*(n-m-v)h_2^*(v)\right]\\ &= \sum_u\sum_v h_1(u)R_{X_1X_2}(m+v-u)h_2^*(v).\end{aligned}$$

Looking at the sum over u, we recognize a convolution product between h_1 and $R_{X_1X_2}$, expressed at time $(m+v)$:

$$\begin{aligned}R_{Y_1Y_2}(m) &= \sum_v (h_1 * R_{X_1X_2})(m+v)h_2^*(v)\\ &= \sum_v (h_1 * R_{X_1X_2})(m+v)h_2^{(-)*}(-v),\end{aligned}$$

where we have noted $h_2^{(-)}(v) = h_2(-v)$. In this last relation, we recognize anoter convolution product, this time between $(h_1 * R_{X_1X_2})$ and $h_2^{*(-)}$:

$$\begin{aligned}R_{Y_1Y_2}(m) &= \sum_v (h_1 * R_{X_1X_2})(m+v)h_2^{*(-)}(-v)\\ &= \sum_{v'} (h_1 * R_{X_1X_2})(m-v')h_2^{*(-)}(v')\\ &= \left(h_1 * R_{X_1X_2} * h_2^{*(-)}\right)(m).\end{aligned}$$

We finally obtain the important formula:

$$\boxed{R_{Y_1Y_2}(m) = \left(h_1 * R_{X_1X_2} * h_2^{*(-)}\right)(m)}. \tag{1.25}$$

1.4.2 By-products

- **[Autocorrelation of the output of a filter]** With a single filter we can apply the previous formula, with

$$\begin{cases} X_1 = X_2 = X, \\ h_1 = h_2 = h. \end{cases} \tag{1.26}$$

Of course $Y_1 = Y_2 = Y$, and

$$\boxed{R_{YY}(m) = \left(h * R_{XX} * h^{*(-)}\right)(m)}. \tag{1.27}$$

- **[Cross correlation between output and input]** We want to measure the correlation between the input and output of a filter. Toward this goal, we consider

$$\begin{cases} X_1 = X_2 = X, \\ Y_1 = Y, \\ Y_2 = X, h_1 = h, \\ h_2 = h. \end{cases} \tag{1.28}$$

In this case, we got

$$\boxed{R_{YX}(m) = (h * R_{XX})(m)}. \tag{1.29}$$

The cross correlation between the output and the input of filter has the very same apparence as the filtering which links the output to the input.

1.4.3 Examples

We study now the filtering of random signals. We begin with the classical impulse response $h(n) = a^n$, with $x(n)$ a uniform distributed white noise at the input, and we denote $y(n)$ the output.

1. Filter the signal $x(n)$, with the help of the function `lfilter`. Compare the input and output signals, and in particular their variations. Compare the histograms. Look at the Fourier transform of the output. Do this for several values of a, beginning with $a = 0.9$.

2. Using the function `xcorr` (import it via `from correlation import xcorr`), compute all the possible correlations between the input and the output. What would be the correlation matrix associated with the signal $x(n) = [x(n)\ y(n)]^t$? Compare the impulse response h to the cross-correlation $Ryx(k)$. Explain. Experiment by varying the number of samples N and a (including its sign).

3. Consider the identification of the impulse response by cross-correlation, as above, but in the noisy case. Add a Gaussian noise to the output and compute the cross-correlation. Observe, comment and experiment with the parameters.

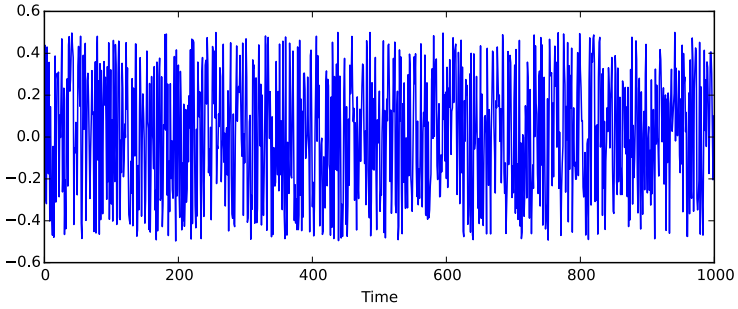
The filtering is done thanks to the function `lfilter`. We have first to import it, *eg* as

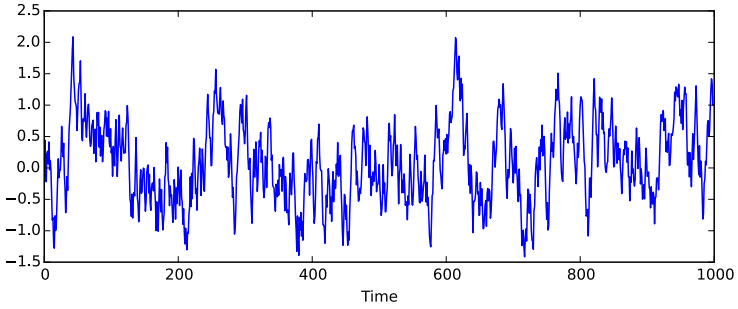
```
from scipy.signal import lfilter
```

We will also need to play with ffts so it is a good time to import it from fftpack

```
from scipy.fft import fft, ifft
```

```
N=1000 #Number of samples
x=stats.uniform(-0.5,1).rvs(N)
a=0.9
y=lfilter([1],[1, -a],x)
figure(figsize=(8,3))
plot(x); xlabel("Time")
figure(figsize=(8,3))
plot(y); xlabel("Time")
```



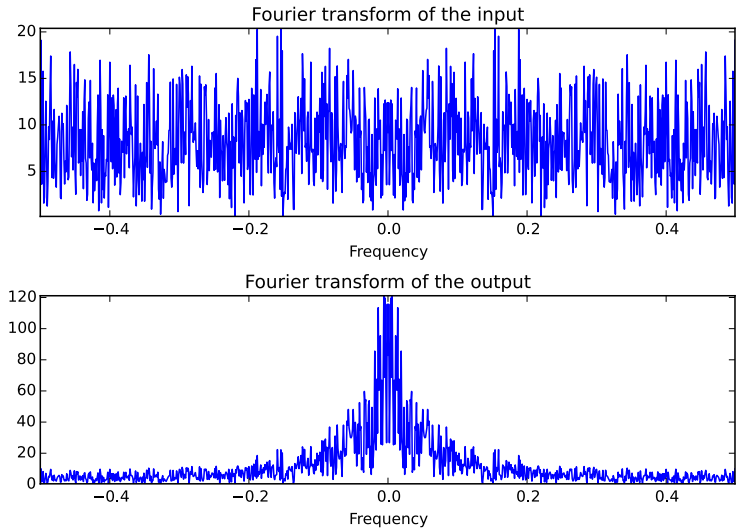


We see that the output has slower variations than the input. This is the result of the filtering operation.

Let us now look at the Fourier transform:

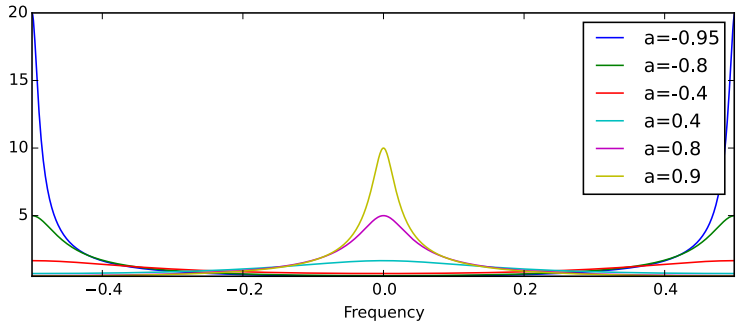
```
f=arange(N)/N-0.5
fig,ax=subplots(2,1,figsize=(7,5))
ax[0].plot(f,abs(fftshift(fft(x))))
ax[0].set_title("Fourier transform of the input")
ax[0].set_xlabel("Frequency")
ax[0].axis('tight') #Tight layout of the axis
ax[1].plot(f,abs(fftshift(fft(y))))
ax[1].set_title("Fourier transform of the output")
ax[1].set_xlabel("Frequency")
fig.tight_layout()
ax[1].axis('tight')
```

```
(-0.5, 0.499, 0.36038862332018706, 121.13277088361765)
```



Recall how the transfer function changes with a:

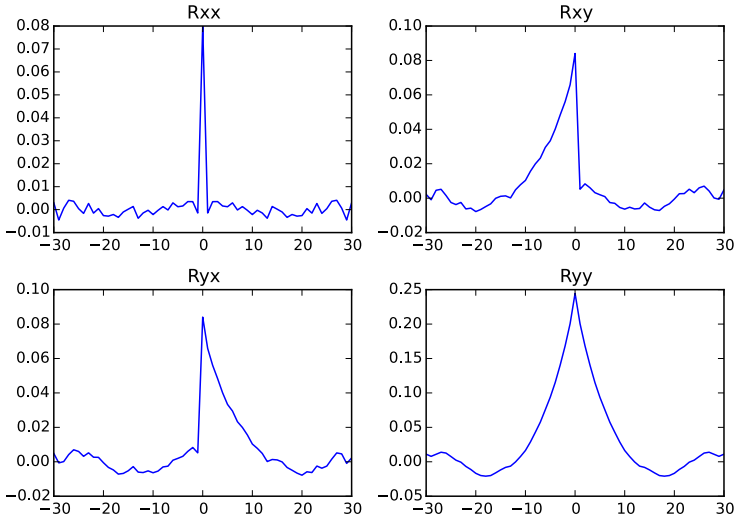
```
figure(figsize=(8,3))
d=zeros(N); d[0]=1
for a in (-0.95, -0.8, -0.4, 0.4, 0.8, 0.9):
    h=lfilter([1],[1, -a],d)
    H=fft(h)
    plot(f,abs(fftshift(H)),label='a={}'.format(a))

legend()
axis('tight')
_=xlabel("Frequency")
```



1.4.4 Correlation matrix

```
from correlation import xcorr
N=1000 #Number of samples
x=stats.uniform(-0.5,1).rvs(N)
a=0.8
y=lfilter([1],[1, -a],x)

L=30
Rxx, lags=xcorr(x,x,maxlags=L)
```

```
Rxy, lags=xcorr(x,y,maxlags=L)
Ryx, lags=xcorr(y,x,maxlags=L)
Ryy, lags=xcorr(y,y,maxlags=L)
fig ,ax=subplots(2,2,figsize=(7,5))
axf=ax.flatten()
Rtitles=('Rxx','Rxy','Ryx','Ryy')
for k,z in enumerate((Rxx,Rxy,Ryx,Ryy)):
    axf[k].plot(lags,z)
    axf[k].set_title(Rtitles[k])
fig.tight_layout()
```



We have represented above all the possible correlations between the input and the ouput. This representation corresponds to the correlation matrix of the vector $z(n) = [x(n) \;\; y(n)]^H$ that would give

$$\mathbb{E}\left\{ \begin{bmatrix} x(n) \\ y(n) \end{bmatrix} \begin{bmatrix} x(n-k)^* & y(n-k)^* \end{bmatrix} \right\} = \begin{bmatrix} R_{xx}(k) & R_{xy}(k) \\ R_{yx}(k) & R_{yy}(k) \end{bmatrix} \tag{1.30}$$

1.4.5 Identification of a filter by cross-correlation

We know that the cross-correlation of the output of a system with IR h with its input is given by

$$R_{yx}(k) = (R_{xx} * h)(k). \tag{1.31}$$

When the input is a white noise, then its autocorrelation $R_{xx}(k)$ is a Dirac impulse with weight σ_x^2, $R_{xx}(k) = \sigma_x^2 \delta(k)$, and R_{yx} is proportional to the impulse response:

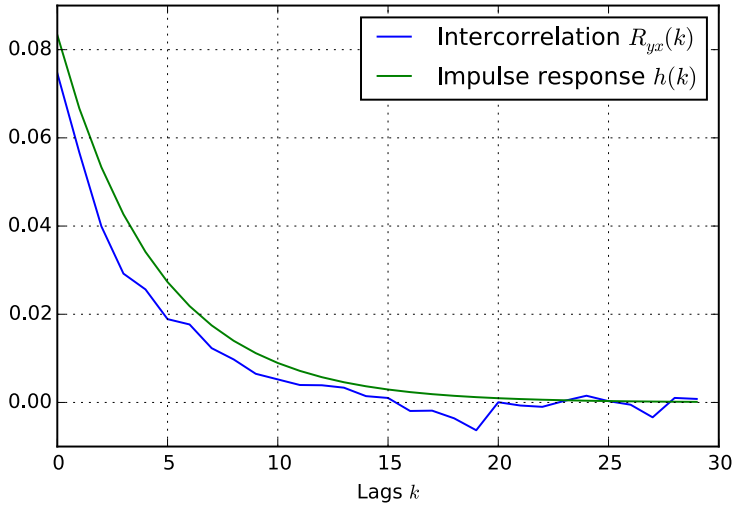
$$R_{yx}(k) = \sigma_x^2 h(k). \tag{1.32}$$

This is what we observe here:

```
# An object "uniform random variable" with fixed bounds [0,1]
x_uni=stats.uniform(loc=0,scale=1)
(m,v)=x_uni.stats(moments='mv')
print("Uniform distribution: ","Value of the mean : {0:2.3f} and of the
    variance {1:2.3f}".format(float(m),float(v)))
N=1000 #Number of samples
x=stats.uniform(-0.5,1).rvs(N)  #generates N values for x
a=0.8
```

```
y=lfilter([1],[1, -a],x) #Computes the output of the system
L=30
Ryx,lags=xcorr(y,x,maxlags=L) #then the cross-correlation
d=zeros(N); d[0]=1
h=lfilter([1],[1, -a],d)  #and the impulse response
plot(arange(L),Ryx[arange(L,L+L)],label="Intercorrelation $R_{yx}(k)$")
plot(arange(L),v*h[arange(L)],label="Impulse response $h(k)$")
xlabel("Lags $k$")
grid(True)
legend()
```

```
Uniform distribution:  Value of the mean : 0.500 and of the variance 0.083
```



In the noisy case, the same kind of observations hold. Indeed, if z is a corrupted version of y, with $z(n) = y(n) + w(n)$, then

$$R_{zx}(k) = R_{yx}(k) + R_{wx}(k) = R_{yx}(k) \tag{1.33}$$

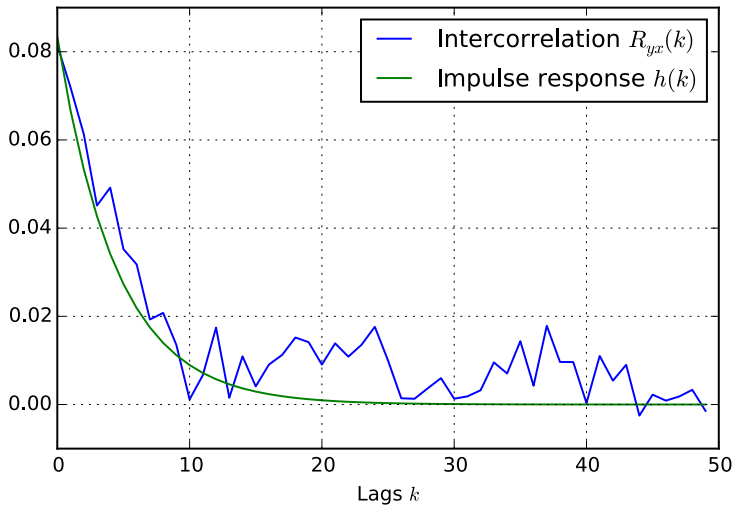
provided that x and w are uncorrelated, which is reasonable assumption.

```
N=1000
#Remember that the variance of $x$ is given by
x_uni=stats.uniform(-0.5,1)
(m,v)=x_uni.stats(moments='mv')
print("Uniform distribution: ","Value of the mean : {0:2.3f} and of the
    variance {1:2.3f}".format(float(m),float(v)))
```

```
Uniform distribution:  Value of the mean : 0.000 and of the variance 0.083
```

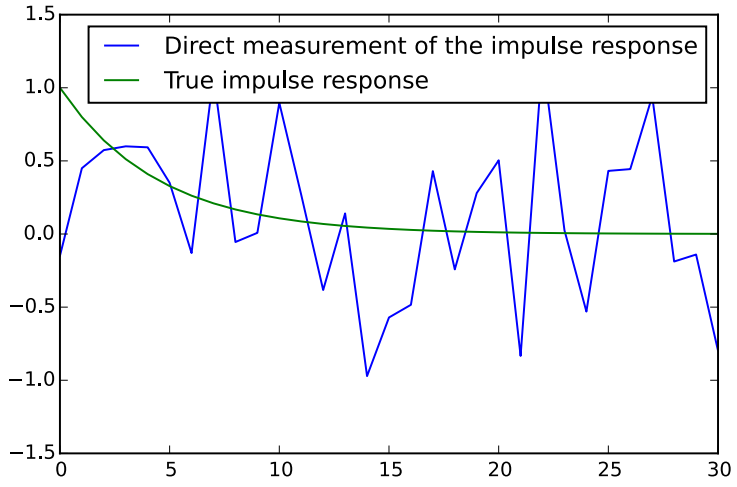
```
N=1000 #Number of samples
x=stats.uniform(-0.5,1).rvs(N)  #generates N values for x
a=0.8
y=lfilter([1],[1, -a],x)  #Computes the output of the system
w=stats.norm(0,1).rvs(N)  #Gaussian noise
y=y+0.5*w
L=50
Ryx,lags=xcorr(y,x,maxlags=L) #then the cross-correlation
```

```
d=zeros(N); d[0]=1
h=lfilter([1],[1, -a],d)  #and the impulse response
plot(arange(L),Ryx[arange(L,L+L)],label="Intercorrelation $R_{yx}(k)$")
plot(arange(L),v*h[arange(L)],label="Impulse response $h(k)$")
xlabel("Lags $k$")
grid(True)
legend()
```



while the direct measure of the impulse response would give

```
plot(arange(N),h+0.5*w,label="Direct measurement of the impulse response")
plot(arange(N),h,label="True impulse response")
xlim([0, 30])
_=legend()
```



Hence, we see that identification of a system is possible by cross-correlation, even when dealing with noisy outputs. Such identification would be impossible by direct measurement of the IR, because of the presence of noise.

1.5 Analyse dans le domaine fréquentiel

En repartant de la formule des interférences

$$R_{Y_1Y_2}(m) = \left(h_1 * R_{X_1X_2} * h_2^{*(-)}\right)(m), \tag{1.34}$$

on obtient simplement, après transformée de Fourier,

$$\boxed{S_{Y_1Y_2}(f) = H_1(f)S_{X_1X_2}(f)H_2^*(f)}, \tag{1.35}$$

où $S_{Y_1Y_2}(f), S_{X_1X_2}(f), H_1(f)$ et $H_2(f)$ sont respectivement les transform'ees de Fourier de $R_{Y_1Y_2}(m)$, $R_{X_1X_2}(m)$, $h_1(m)$ et $h_2(m)$. Note{La transformée de Fourier de $h^*(-n)$ vaut $H^*(f)$.}\
Conséquences :

1. En prenant $Y_1 = Y_2 = Y$, $X_1 = X_2 = X$ et $H_1 = H_2 = H$, c'est-à-dire que l'on considère un seul filtre, il vient
$$\boxed{S_{YY}(f) = S_{XX}(f)|H(f)|^2}. \tag{1.36}$$
2. Si $H_1(f)$ et $H_2(f)$ sont deux filtres *disjoints* en fréquence, alors
$$S_{Y_1Y_2}(f) = 0. \tag{1.37}$$
On en déduit que
$$R_{Y_1Y_2}(\tau) = \mathrm{TF}^{-1}[S_{Y_1Y_2}(f)] = \mathrm{TF}^{-1}[0] = 0. \tag{1.38}$$
si les filtres sont disjoints en fréquence, l'intercorrélation des sorties est nulle.

Application Considérons deux filtres parfaits autour de deux fréquences pures f_1 et f_2, de même entrée $X(n,\omega)$. On a $Y_1(n,\omega) = X(f_1,\omega)\exp(-j2\pi f_1 n)$, et $Y_2(n,\omega) = X(f_2,\omega)\exp(-j2\pi f_2 n)$, avec toutes les précautions d'usage sur la << non existence >> de la transformée de Fourier considérée pour des signaux aléatoires stationnaires. Dans ces conditions,

$$R_{Y_1Y_2}(0) = \mathbb{E}\left[X(f_1,\omega)X^*(f_2,\omega)\right]\exp(-j2\pi(f_1-f_2)n) = 0, \tag{1.39}$$

soit

$$\mathbb{E}\left[X(f_1,\omega)X^*(f_2,\omega)\right] = 0. \tag{1.40}$$

On dit que les composantes spectrales sont décorrélées.

1.5.1 Notion de densité spectrale de Puissance

La densité spectrale de puissance représente la répartition de la puissance du signal dans le domaine fréquentiel. Il s'agit exactement de la même notion que celle de densité de probabilité : lorsque l'on veut calculer probabilité qu'une variable aléatoire X appartienne à un certain intervalle $[x_1, x_2]$, il suffit d'intégrer la densité de probabilité de la variable entre ces deux bornes :

$$\Pr(X \in [x_1, x_2]) = \int_{x_1}^{x_2} p(X)\mathrm{d}X. \tag{1.41}$$

Si on appelle $D_{XX}(f)$ la densité spectrale de puissance d'un signal aléatoire $X(n,\omega)$, alors la puissance du signal portée par les composantes fréquentielles comprises entre f_1 et f_2 s'écrit

$$P_{XX}(f \in [f_1, f_2]) = \int_{f_1}^{f_2} D_{XX}(f)\mathrm{d}f. \tag{1.42}$$

Dès lors, la puissance totale du signal s'écrit

$$P_{XX} = \int_{-\frac{1}{2}}^{+\frac{1}{2}} D_{XX}(f)\mathrm{d}f. \tag{1.43}$$

Or on sait que, pour un signal stationnaire et ergodique,

$$P_{XX} = \mathbb{E}\left[|X(n,\omega)|^2\right] = R_{XX}(0) = \lim_{N\to+\infty} \frac{1}{2N} \sum_{-N}^{+N} |X(n,\omega)|^2. \tag{1.44}$$

Par ailleurs,

$$R_{XX}(\tau) = \int_{-\frac{1}{2}}^{+\frac{1}{2}} S_{XX}(f) \exp\left(j2\pi f\tau\right) \mathrm{d}f, \tag{1.45}$$

soit, pour $\tau = 0$,

$$R_{XX}(0) = P_{XX} = \int_{-\frac{1}{2}}^{+\frac{1}{2}} S_{XX}(f)\ \mathrm{d}f. \tag{1.46}$$

La transformée de Fourier $S_{XX}(f)$ de la fonction d'autocorrélation est ainsi une bonne candidate pour être la densité spectrale de puissance. Notons cependant, cette dernière relation ne prouve pas qu'elle le soit.

Considérons un filtre parfait, dont le module de la fonction de transfert est d'amplitude un dans une bande Δf centrée sur une fréquence f_0, et nul ailleurs :

$$\begin{cases} |H(f)| = 1 \text{ pour } f \in [f_0 - \frac{\Delta f}{2}, f_0 + \frac{\Delta f}{2}] \\ |H(f)| = 0 \text{ ailleurs.} \end{cases} \tag{1.47}$$

Notons $Y(n,\omega) = (h * X)(n,\omega)$ la réponse de ce filtre à une entrée $X(n,\omega)$. La puissance de la sortie est donnée par

$$P_{YY} = R_{YY}(0) = \int_{-\frac{1}{2}}^{+\frac{1}{2}} S_{YY}(f)\ \mathrm{d}f, \tag{1.48}$$

or la formule des interférences fournit

$$S_{YY}(f) = S_{XX}(f)|H(f)|^2, \tag{1.49}$$

avec les conditions sur le module de $H(f)$ données précédemment. On obtient donc

$$P_{YY}(f \in [f_0 - \frac{\Delta f}{2}, f_0 + \frac{\Delta f}{2}]) = \int_{f_0-\frac{\Delta f}{2}}^{f_0+\frac{\Delta f}{2}} S_{XX}(f)\ \mathrm{d}f, \tag{1.50}$$

ce qui correspond bien à la définition de la densité spectrale de puissance : la puissance pour les composantes spectrales comprises dans un intervalle est bien égale à l'intégrale de la densité spectrale de puissance sur cet intervalle. Si Δf est suffisamment faible, on pourra considérer la densité spectrale de puissance $S_{XX}(f)$ comme approximativement constante sur l'intervalle, et

$$P_{YY}(f \in [f_0 - \frac{\Delta f}{2}, f_0 + \frac{\Delta f}{2}]) \simeq S_{XX}(f_0)\Delta f. \tag{1.51}$$

Cette dernière relation indique que la densité spectrale de puissance doit s'exprimer en Watts par Hertz. Par ailleurs, lorsque Δf tend vers 0, la puissance recueillie est de plus en plus faible. Pour

$\Delta f = 0$, la puissance obtenue est ainsi normalement nulle, sauf si la densité spectrale elle-même est constituée par une << masse >> de Dirac (de largeur nulle mais d'amplitude infinie) à la fréquence considérée.

Notons que le filtre que nous avons défini ci-dessus n'est défini, par commodité de présentation, que pour les fréquences positives. Sa fonction de transfert ne vérifie donc pas la propriété de symétrie hermitienne des signaux réels : la réponse impulsionnelle associée est donc complexe et la sortie $Y(t,\omega)$ également complexe. En restaurant cette symétrie, c'est-à-dire en imposant $H(f) = H^*(-f)$, ce qui entraîne (notez le module de f)

$$\begin{cases} |H(f)| = 1 \text{ pour } |f| \in [f_0 - \frac{\Delta f}{2}, f_0 + \frac{\Delta f}{2}] \\ |H(f)| = 0 \text{ ailleurs,} \end{cases} \tag{1.52}$$

la puissance en sortie est

$$P_{YY} = \int_{-f_0-\frac{\Delta f}{2}}^{-f_0+\frac{\Delta f}{2}} S_{XX}(f)\ \mathrm{d}f + \int_{f_0-\frac{\Delta f}{2}}^{f_0+\frac{\Delta f}{2}} S_{XX}(f)\ \mathrm{d}f. \tag{1.53}$$

La densité spectrale de puissance d'un signal aléatoire réel est une fonction paire, ce qui conduit enfin à

$$P_{YY} = 2\int_{f_0-\frac{\Delta f}{2}}^{f_0+\frac{\Delta f}{2}} S_{XX}(f)\ \mathrm{d}f, \tag{1.54}$$

relation qui indique que la puissance se partage équitablement dans les fréquences positives et négatives.\

Exemple :\

Considérons le cas d'une sinusoïde à amplitude et phase aléatoire

$$X(n,\omega) = A(\omega)\sin(2\pi f_0 n + \phi(\omega)), \tag{1.55}$$

où $A(\omega)$ est une variable aléatoire centrée de variance σ^2 et $\phi(\omega)$ uniformément répartie sur $[0, 2\pi]$. La fonction d'autocorrélation de ce signal vaut

$$R_{XX}(\tau) = \frac{\sigma^2}{2}\cos(2\pi f_0 \tau). \tag{1.56}$$

Par transformée de Fourier, on obtient la densité spectrale :

$$S_{XX}(f) = \frac{\sigma^2}{4}[\delta(f+f_0) + \delta(f-f_0)]. \tag{1.57}$$

Enfin, en intégrant la densité spectrale

$$\int \frac{\sigma^2}{4}[\delta(f+f_0) + \delta(f-f_0)]\mathrm{d}f = \frac{\sigma^2}{2}, \tag{1.58}$$

on retrouve la puissance de la sinusoïde, $\sigma^2/2$, comme il se doit.\

Les fonctions de corrélation et les densités spectrales de puissance forment des paires de transformées de Fourier :

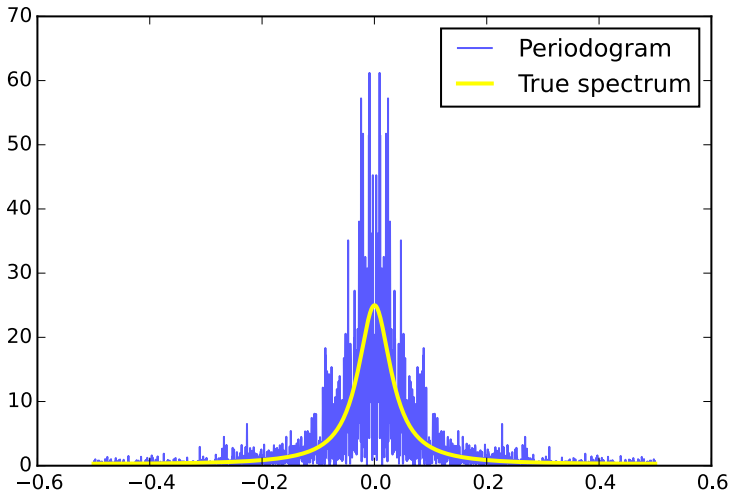
$$\boxed{\begin{array}{rcl} S_{XX}(f) & \rightleftharpoons & R_{XX}(\tau), \\ S_{XY}(f) & \rightleftharpoons & R_{XY}(\tau), \end{array}} \tag{1.59}$$

où $S_{XX}(f)$, $S_{XY}(f)$ sont les densités spectrale de puissance et de puissance d'interaction, respectivement. Ces relations constituent le **théorème de Wiener-Kintchine-Einstein**.

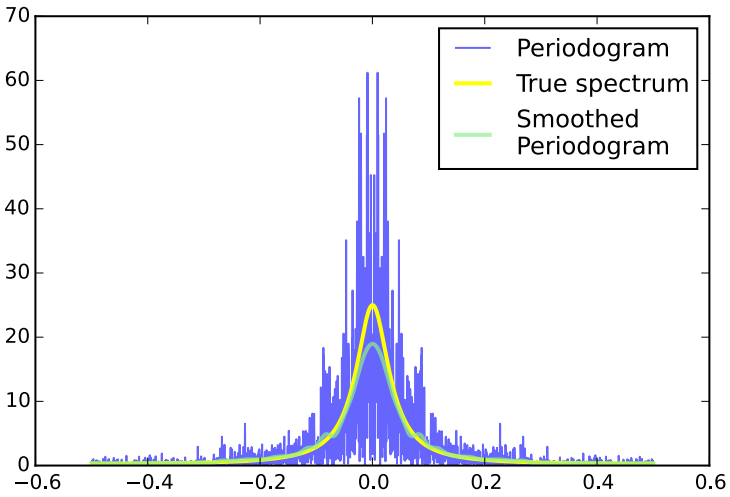
1.5.2 Power spectrum estimation

```
from scipy.signal import lfilter
from numpy.fft import fft, ifft, fftshift, fftfreq
```

```
N=2000
a=-0.8
x=stats.norm(0,1).rvs((N))
y=lfilter([1],[1, a],x)
Yf=fft(y)
Py=1/N*abs(Yf)**2
f=fftfreq(N)
f=np.linspace(-0.5,0.5,N)
Sy=abs(1/abs(fft([1, a],N))**2)
plt.plot(f,fftshift(Py),alpha=0.65,label="Periodogram")
plt.plot(f,fftshift(Sy),color="yellow", lw=2,label="True spectrum")
plt.legend()
#
# Smoothing
#
Ry=ifft(Py)
hh=sig.hamming(200,sym=True)
```

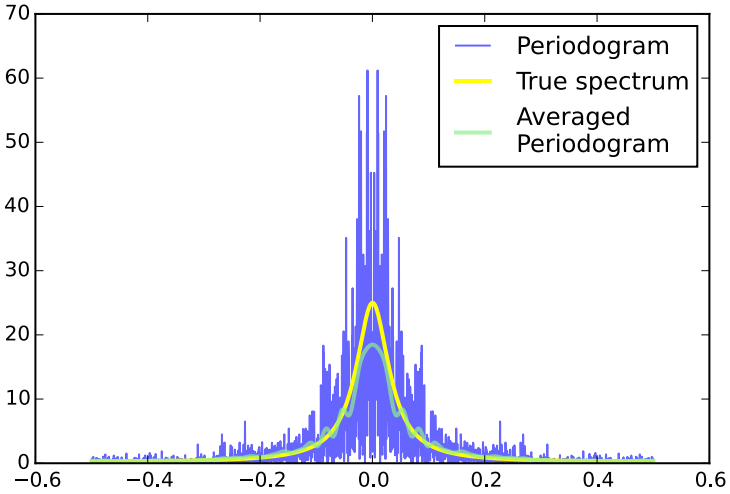



```
z=np.zeros(N)
L=100
h=fftshift(sig.hanning(L,sym=True))
z[0:round(L/2)]=h[0:round(L/2)]
z[-1:-round(L/2)-1:-1]=h[-1:-round(L/2)-1:-1]
Py_smoothed=abs(fft(z*Ry))
plt.plot(f,fftshift(Py),alpha=0.6,label="Periodogram")
plt.plot(f,fftshift(Sy),lw=2,color="yellow",label="True spectrum")
plt.plot(f,fftshift(Py_smoothed),alpha=0.7,color="lightgreen", lw=2,label=
    "Smoothed \nPeriodogram")
_=plt.legend()
```




```
#Averaging
def averaged_perio(y,M):
    N=np.size(y)
    L=np.round(N/M)
    Py_averaged=np.zeros(N)
    for m in range(M):
        Py_averaged+=1/L*(abs(fft(y[m*L:(m+1)*L],N))**2)
    return  Py_averaged/M

Py_averaged=averaged_perio(y,40)

plt.plot(f,fftshift(Py),alpha=0.6,label="Periodogram")
plt.plot(f,fftshift(Sy),lw=2,color="yellow",label="True spectrum")
plt.plot(f,fftshift(Py_averaged),alpha=0.7,color="lightgreen",lw=2,label="
    Averaged \nPeriodogram")
plt.legend()
```

```
/usr/local/lib/python3.4/dist-packages/ipykernel/__main__.py:7: DeprecationWarning: using a non-
```



1.6 Applications

1.6.1 Matched filter

We consider a problem frequently encountered in practice, in applications as echography, seismic reflexion, sonar or radar. The problem at hand is as follows: we look for a *known waveform* $s(n)$, up to a delay n_0 in a mixture

$$y(n) = As(n - n_0) + v(n), \tag{1.60}$$

where A and n_0 are unknowns and $v(n)$ is an additive noise. The problem is to find the delay n_0, which typically corresponds to a time-to-target. In order to do that, supppose that we filter the mixture by a filter with impulse response h. The ouput has the form

$$z(n) = x(n) + w(n), \tag{1.61}$$

with $x(n) = A[h * s](n - n_0)$ and $w(n) = [h * v](n)$, respectively the outputs of the signal and noise part. Clearly, if $v(n)$ is stationnary, so is $w(n)$. Therefore, the idea is to design h so that the signal output is as greater as possible than the noise output, at time n_0. In statistical terms, we put this as choosing the filter such that ratio of the signal output'*power* to the noise output's power is maximum. Hence, our goal is to design a filter which maximizes the signal-to-noise ratio at time n_0. We suppose that the desired signal is deterministic and thus consider its instantaneous power $|x(n_0)|^2$.

The signal-to-noise ratio at time n_0 is

$$SNR(n_0) = \frac{|x(n_0)|^2}{\mathbb{E}\left[|w(n)|^2\right]}. \tag{1.62}$$

Of course, both the numerator and the denominator depends on the filter. Lets us first consider the numerator. We have

$$\begin{aligned} x(n_0) &= \mathrm{FT}^{-1}[X(f)]_{n=n_0} && (1.63)\\ &= \int H(f)\mathrm{FT}[s(n-n_0)]e^{j2\pi fn}\mathrm{d}f\Big|_{n=n_0} && (1.64)\\ &= \int H(f)S(f)e^{-j2\pi fn_0}e^{j2\pi fn}\mathrm{d}f\Big|_{n=n_0} && (1.65)\\ &= \int H(f)S(f)\mathrm{d}f. && (1.66) \end{aligned}$$

As far as the denominator is concerned, we have by the Wiener-Kintchine theorem, that

$$\mathbb{E}\left[|w(n)|^2\right] = \int S_{WW}(f)\mathrm{d}f = \int |H(f)|^2 S_{VV}(f)\mathrm{d}f. \tag{1.67}$$

Finally, the signal-to-noise ratio becomes

$$\boxed{SNR(n_0) = \frac{\left|\int H(f)S(f)\mathrm{d}f\right|^2}{\int |H(f)|^2 S_{VV}(f)\mathrm{d}f}.} \tag{1.68}$$

In order to maximize the signal-to-noise ratio we invoke the Cauchy-Schwarz inequality. Recall that that this inequality states that given to integrable functions f and g and a positive measure w, then

$$\left|\int f(x)g(x)^* w(x)\mathrm{d}x\right|^2 \leq \int |f(x)|^2 w(x)\mathrm{d}x \int |g(x)|^2 w(x)\mathrm{d}x \tag{1.69}$$

with equality if and only if $f(x) = kg(x)$ for any arbitrary real constant k.

The idea is to apply this inequality in order to simplify the $SNR(n_0)$. For that, let us express the numerator as

$$\int H(f)S(f)\mathrm{d}f = \int H(f)\sqrt{S_{VV}(f)}\frac{S(f)}{\sqrt{S_{VV}(f)}}\mathrm{d}f. \tag{1.70}$$

By the Cauchy-Schwarz inequality, we then get that

$$\left|\int H(f)S(f)\mathrm{d}f\right|^2 \leq \int |H(f)|^2 S_{VV}(f)\mathrm{d}f \int \left|\frac{S(f)}{\sqrt{S_{VV}(f)}}\right|^2 \mathrm{d}f \tag{1.71}$$

Injecting this inequality in the $SNR(n_0)$ we obtain that

$$\boxed{SNR(n_0) \leq \int \left|\frac{S(f)}{\sqrt{S_{VV}(f)}}\right|^2 \mathrm{d}f}. \tag{1.72}$$

This shows that the SNR at n_0 **is upper bounded** by a quantity which is independent of $H(f)$. Furthermore, by the conditions for equality in the Cauchy-Schwartz inequality, we have that the bound is attained if and only if

$$H(f) = k\frac{S(f)^*}{S_{VV}(f)}. \tag{1.73}$$

In the special case where $v(n)$ is a white, then $S_{VV}(f)$ is a constant, say $S_{VV}(f) = \sigma^2$, and

$$H(f) = k'S(f)^*. \tag{1.74}$$

By inverse Fourier transform, the corresponding impulse response is nothing but

$$h(n) = k's(-n)^*, \tag{1.75}$$

that is, the **complex conjugate and reversed** original waveform. This will be important to link the output of the filter to an estimate of the cross-correlation function. For now, let us also observe that the general transfer function $H(f)$ can be interpreted as a a **whitening operation** followed by the matched filter for an additive white noise:

$$H(f) = k\frac{S(f)^*}{S_{VV}(f)} = k\underbrace{\frac{1}{\sqrt{S_{VV}(f)}}}_{\text{whitening}} \times \underbrace{\frac{S(f)^*}{\sqrt{S_{VV}(f)}}}_{\text{matched filter}} \tag{1.76}$$

Finally, the output of the matched filter can be viewed as the computation of an estimated of the cross-correlation function. Indeed, the output of the $h(n)$ with input x is

$$y(n) = \sum_l h(l)x(n-l) \tag{1.77}$$

$$= \sum_l s(-l)^* x(n-l) \tag{1.78}$$

$$= \sum_m s(m)^* x(n+m) \tag{1.79}$$

$$= \hat{R}_{xs}(n), \tag{1.80}$$

where $\hat{R}_{xs}(n)$ is, up to a factor, an estimate of the cross-correlation between x and s. Applying this remark to our initial mixture

$$y(n) = As(n - n_0) + v(n) \tag{1.81}$$

we get that

$$z(n) = A\hat{R}_{ss}(n - n_0) + \hat{R}_{vs}(n). \tag{1.82}$$

Finally, since v and s are uncorrelated, $\hat{R}_{vs}(n) \simeq 0$ and since $\hat{R}_{ss}(n)$ is maximum at zero, we see that the output will present a peak at $n = n_0$, thus enabling to locate the value of the delay n_0.

Matched filter - Experiment

We simulate now a problem in seismic reflection (or in sonar, or radar), where the goal is to detect the positions of interfaces reflecting the incident waves. The time it takes for a reflection from a particular boundary to arrive at the recorder (a geophone) is called the travel time. For a simple vertically traveling wave, the travel time τ from the surface to the reflector and back is called the Two-Way Time (TWT) and is given by the formula $\tau = 2d/c$, with d the distance from the origin to the refector. To a whole set of interfaces then corresponds the observation

$$r(t) = \sum A_i s(t - t_i) + b(t) \tag{1.83}$$

where the t_i are the delays associated with each interface and A_i the reflection coefficients.

In order to localize the interfaces, we use a matched filter, which maximizes the signal to noise ratio.

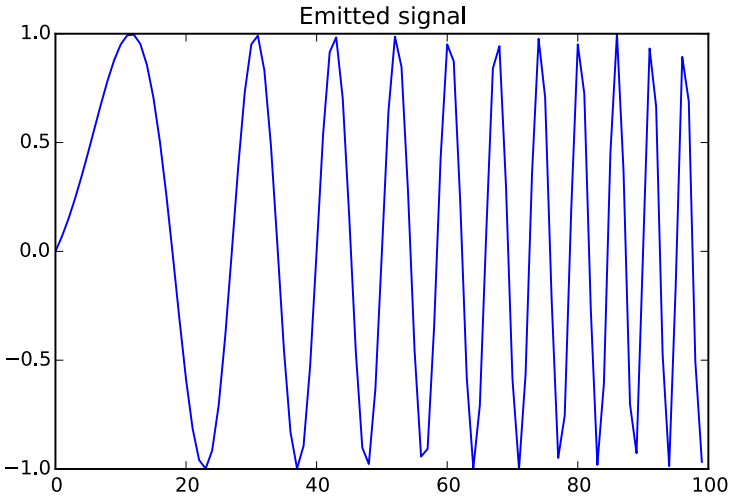
1. Implement the matched filter. Examine the different signals. Is it possible to detect the positions of the interfaces on the time series? using the correlation functions? What is the interest to choose a stimulation signal with a very peaky autocorrelation?

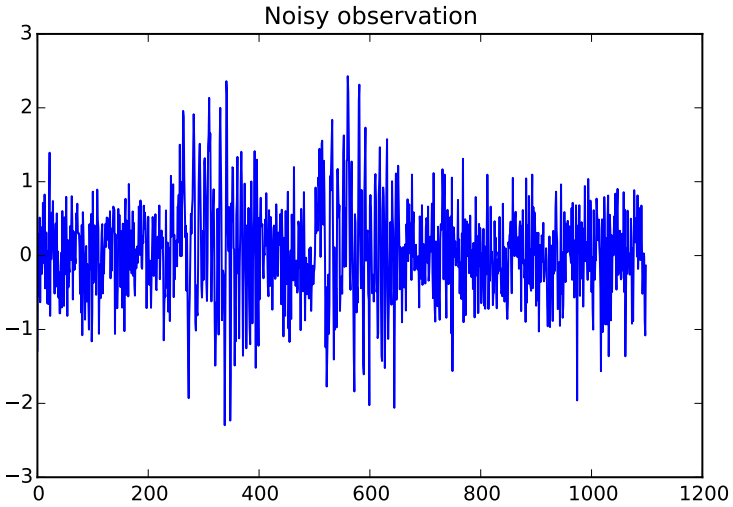
2. Consider a noisy version of the observation (add a Gaussian noise with standard deviation A). Compute the output of the matched filter, with impulse response $h(n) = s(-n)$ and introduce a threshold at 3.3 times the noise standard deviation. Interpret this threshold. Conclusions. Experiment with the level of noise, the number of samples, etc

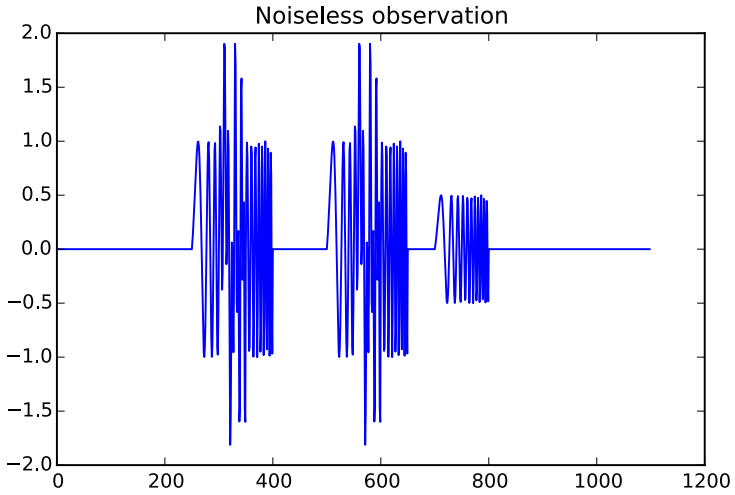
```
def zeropad(v, N):
    a=zeros(N)
    a[arange(len(v))]=v
    return a
```

```
N=1000
#Interface detection by cross-correlation
t=np.arange(100); A=0.5;
s=1*sin(2*pi*0.01*(1+0.1*t)*t) #emitted signal
figure()
plot(t,s)
title('Emitted signal');
```

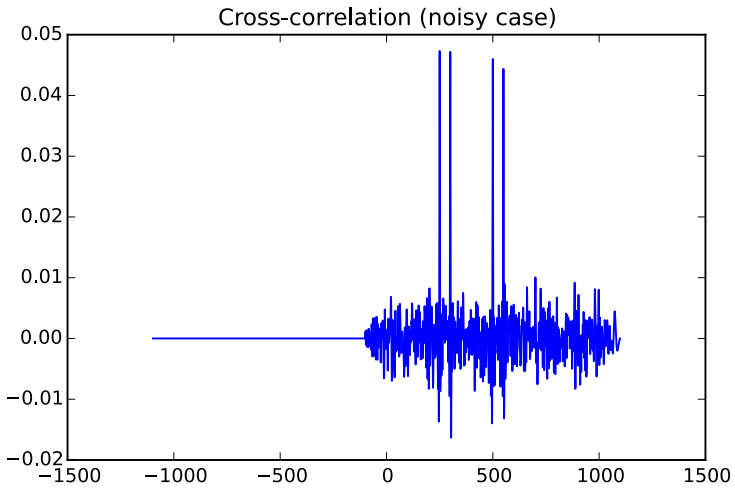
```
# List of interfaces
pos=array([250,300,500,550,700])
amp=array([1,1,1,1,0.5])
g=zeros(N); g[pos]=amp
y=np.convolve(s,g)
z=y+A*randn(size(y))
figure(2); plot(z); title('Noisy observation')
figure(3)
plot(y); title('Noiseless observation')
```

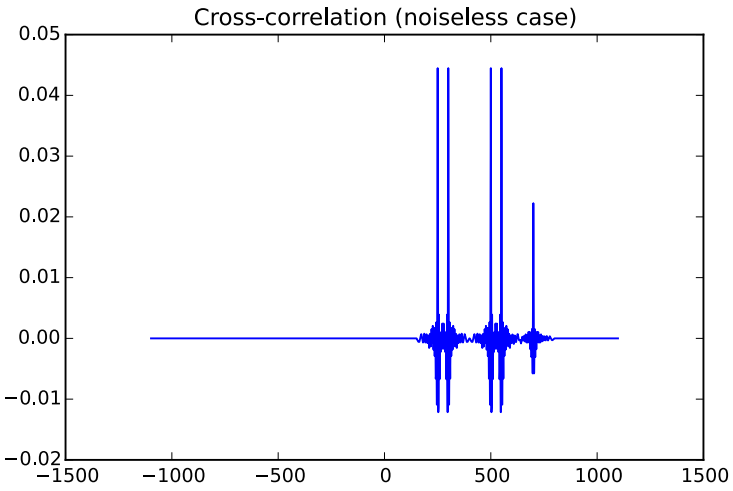







```
from correlation import xcorr
sp=zeropad(s,len(z))
figure(4); Rzs,lags=xcorr(z,sp);
plot(lags,Rzs);
title('Cross-correlation (noisy case)')
figure(5); Rys,lags=xcorr(y,sp);   plot(lags,Rys);
title('Cross-correlation (noiseless case)')
```





Finally, we introduce a threshold in order to eliminate the peaks due to the noise. For that, we compute the threshold so as to have less than some fixed probability to exceed this level.

The method `interval` of an object `stats.norm` returns the endpoints of the range that contains alpha percents of the distribution.

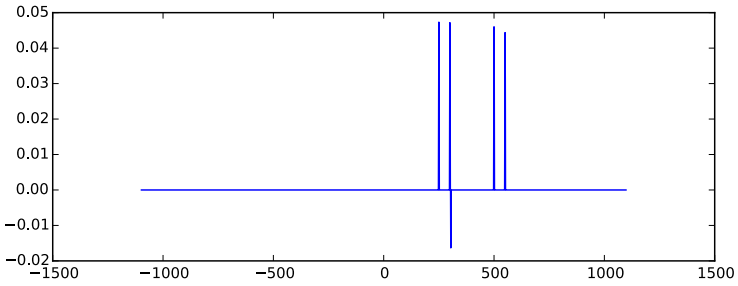
```
interv=stats.norm.interval(alpha=0.999,loc=0,scale=1)
print(interv)
```

```
(-3.2905267314918945, 3.2905267314919255)
```

And the actual thresholding:

```
LR=len(Rzs)
Rzs_th=zeros(LR)
intervs=array(interv)*std(Rzs[500:])

Rzs_th=array([Rzs[u] if (Rzs[u]<intervs[0] or Rzs[u]>intervs[1]) else 0
    for u in range(LR)])
fig,ax=subplots(1,1,figsize=(8,3))
ax.plot(lags,Rzs_th)
print("The position of interfaces are at",where(Rzs_th!=0)[0]+lags[0])
```

```
The position of interfaces are at [249 250 251 299 300 301 304 499 500 501 549 550 551]
```



Quick and Dirty thing to find the “center” of consecutive value ranges

```
def find_center(v):
    Beg=v[0]; Endy=v[0]
    u=0; C=[]
    for k in range(1,len(v)):
        if (v[k]-v[k-1]) in (1,2):
            Endy=Endy+1
        else:
            C.append((Endy+Beg)/2)
            u=u+1
            Beg=v[k]; Endy=v[k]
    if Endy==v[len(v)-1]:
        C.append((Endy+Beg)/2)
    return C
```

```
posit=find_center(where(Rzs_th!=0)[0]+lags[0])
print("Positions where the signal exceeds threshold:\n".ljust(35),
      where(Rzs_th!=0)[0]+lags[0])
print("Detected interfaces positions: ".ljust(35),posit)
print("True positions; ".ljust(35), pos)
```

```
Positions where the signal exceeds threshold:
 [249 250 251 299 300 301 304 499 500 501 549 550 551]
Detected interfaces positions:      [250.0, 300.0, 304.0, 500.0, 550.0]
True positions;                     [250 300 500 550 700]
```

1.6.2 Wiener filtering

Introduction

We consider now the problem of recovering a signal $s(n)$ from an indirect and noisy measurement

$$x(n) = [h * s](n) + v(n). \tag{1.84}$$

This problem involves actually two sub-problems that are very interesting on their own: - *smoothing* of the additive noise, - inversion.

Let us first examine a simple experiment which points-out the necessity of developing a rational approach instead of a adopting a naive one. We generate a random pulse train, filter it, and then reconstruct the input signal by direct division by the transfer function:

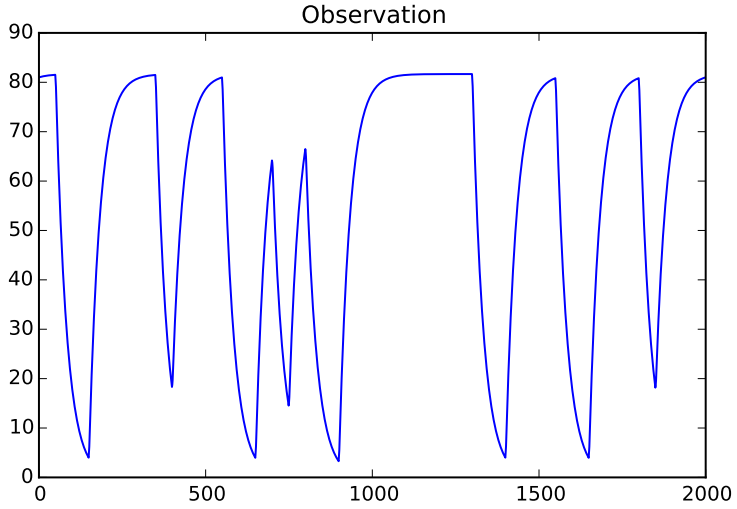
$$S(f) \simeq \frac{X(f)}{H(f)} = S(f) + \frac{V(f)}{H(f)} \tag{1.85}$$

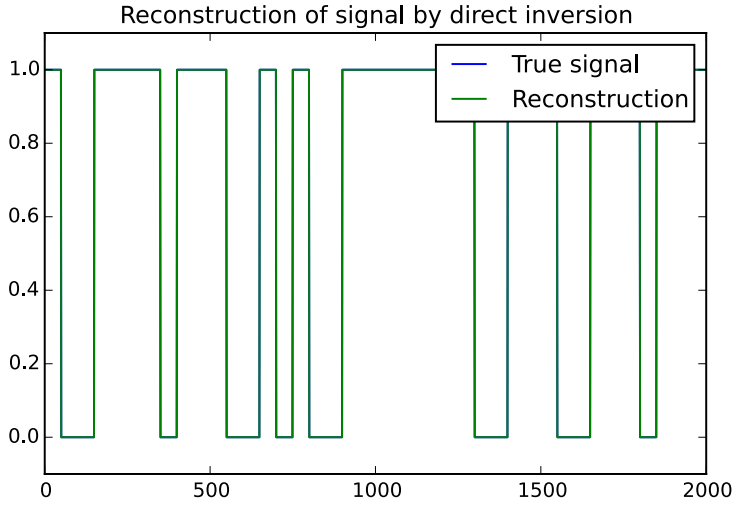
We consider both a noiseless case and a noisy case.

Illustrative experiment

```
N=2000
a=-0.97
L=50
spos=stats.bernoulli.rvs(loc=0,p=0.6,size=N/L)
s=np.kron(spos,np.ones(L))
#x=stats.norm(0,1).rvs((N))
d=np.zeros(N); d[0]=1 #Dirac impulse
h=sig.lfilter([1, 0.5, 0.95],[1, a],d)
#h=sig.lfilter([1, 0.6, 0.95, 1.08, 0.96],[1, a],d)
```

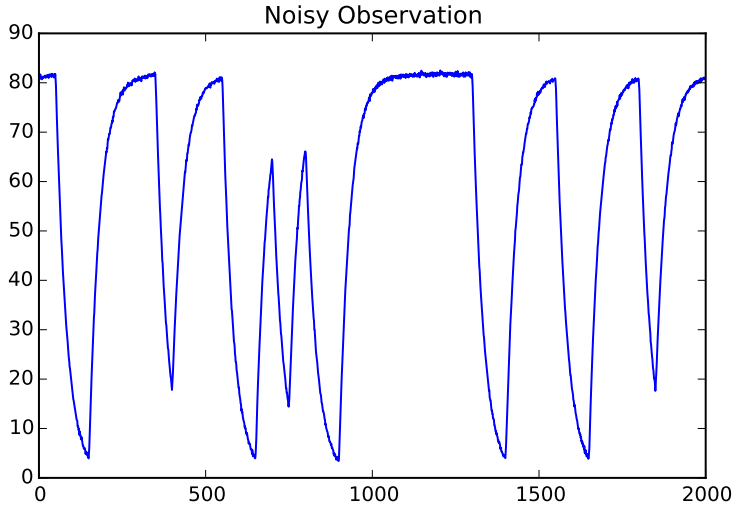
```
H=fft(h,N)
X=fft(s)*H
x=real(ifft(X))

plt.figure()
plt.plot(x)
plt.title("Observation")

#
plt.figure()
x_rec=real(ifft(X/H))
plt.plot(s,label="True signal")
plt.plot(x_rec,label="Reconstruction")
plt.title("Reconstruction of signal by direct inversion")
plt.ylim([-0.1, 1.1])
plt.legend()
```

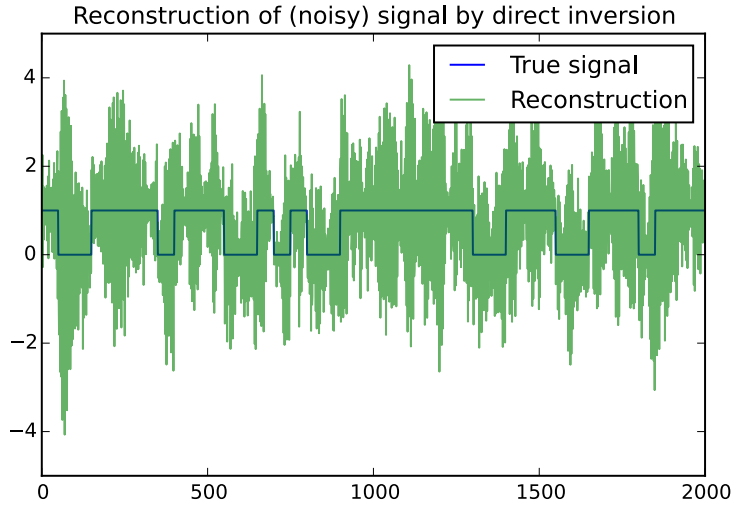





```
# Noisy observation
z=x+0.25*stats.norm(0,1).rvs((N))
Z=fft(z)
```

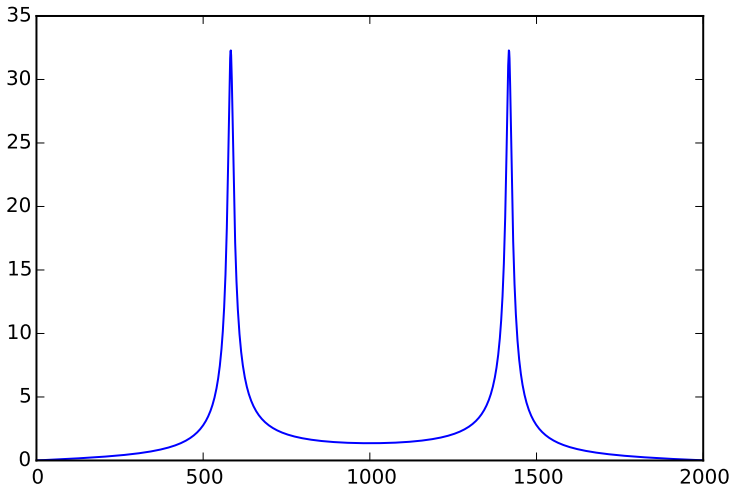
```
plt.figure()
plt.plot(z)
plt.title("Noisy Observation")
plt.figure()
x_rec=real(ifft(Z/H))
plt.plot(s,label="True signal")
plt.plot(x_rec,alpha=0.6,label="Reconstruction")
plt.title("Reconstruction of (noisy) signal by direct inversion")
plt.legend()
```






```
plt.plot(1/abs(H))
```

```
[<matplotlib.lines.Line2D at 0x7f6f55985080>]
```



Derivation of the Wiener filter

Instead of a direct inversion, we put the problem as the design of a filter w which enables to recover an estimate of $s(n)$, from the noisy observation $x(n)$.

$$y(n) = [w * x](n) \tag{1.86}$$

The objective is to minimize the error $e(n) = y(n) - s(n)$, and more precisely of the mean square error

$$\mathbb{E}\left[e(n)^2\right]. \tag{1.87}$$

Recall that

$$\mathbb{E}\left[e(n)^2\right] = R_{EE}[0] = \int S_{EE}(f)\mathrm{d}f. \tag{1.88}$$

Since $e(n) = y(n) - s(n)$, we have that

$$R_{Y-S,Y-S}(k) = R_{YY}(k) - R_{YS}(k) - R_{SY}(k) + R_{SS}(k) \tag{1.89}$$
$$S_{Y-S,Y-S}(f) = S_{YY}(f) - S_{YS}(f) - S_{SY}(f) + S_{SS}(f) \tag{1.90}$$

From the transformation of the power spectrum by filtering and the symmetries of the cross-spectra, we have

$$S_{YY}(f) = |H(f)W(f)|^2 S_{SS}(f) + |W(f)|^2 S_{VV}(f), \tag{1.91}$$
$$S_{YS}(f) = H(f)W(f)S_{SS}(f), \tag{1.92}$$
$$S_{SY}(f) = S_{YS}(f)^*. \tag{1.93}$$

Taking this into account, we arrive at

$$S_{Y-S,Y-S}(f) = |H(f)|^2|W(f)|^2 S_{SS}(f) + |W(f)|^2 S_{VV}(f) + H(f)W(f)S_{SS}(f) + H(f)^*W(f)^*S_{SS}(f) + S_{SS}(f). \tag{1.94}$$

It is easy to check that this formula can be rewritten as

$$S_{Y-S,Y-S}(f) = (S_{SS}(f) + S_{VV}(f))\left|W(f) - \frac{H(f)^* S_{SS}(f)}{|H(f)|^2 S_{SS}(f) + S_{VV}(f)}\right|^2 + S_{SS}(f). \tag{1.95}$$

Clearly, it is minimum if

$$\boxed{W(f) = \frac{H(f)^* S_{SS}(f)}{|H(f)|^2 S_{SS}(f) + S_{VV}(f)}}. \tag{1.96}$$

From this relation, we learn the following: - In the noiseless case, that is $S_{VV}(f) = 0$, then $W(f) = 1/H(f)$. THis is the diert inversion, which is only valid if no noise corrupts the output. - for frequencies where the transfer function $H(f)$ is very small, that is where we have a very small signal part,then $W(f) \sim 0$ (no inversion). - elsewhere, the filter makes a conservative inversion which depends on the local signal-to-noise ratio.

- In the case $H(f) = 1$, the problem reduces to a smoothing problem, that is to suppress the noise without too much corrupting of the signal part. The Wiener filter reduces to

$$\boxed{W(f) = \frac{S_{SS}(f)}{S_{SS}(f) + S_{VV}(f)}}. \tag{1.97}$$

In such case, we see that the transfer function tends to 1 if $S_{SS}(f) \gg S_{VV}(f)$ (frequency bands where the signal is significantly higher than the noise), to zero if $S_{SS}(f) \ll S_{VV}(f)$ (much more noise than signal), and otherwise realises a tradeoff guided by the signal-to-noise ratio in the frequency domain.

Experiment

We consider an example of optimum filtering, the Wiener smoother. Beginning with a noisy mixture $x(n) = s(n) + v(n)$, the goal is to find the best filter which minimizes the noise while preserving the signal: $y(n) = (h * x)(n) \simeq s(n)$.

Simulate a signal

$$s(n) = \exp(-at)\sin(2\pi f_0 t + \phi(\omega)). \tag{1.98}$$

The corresponding implementation lines are

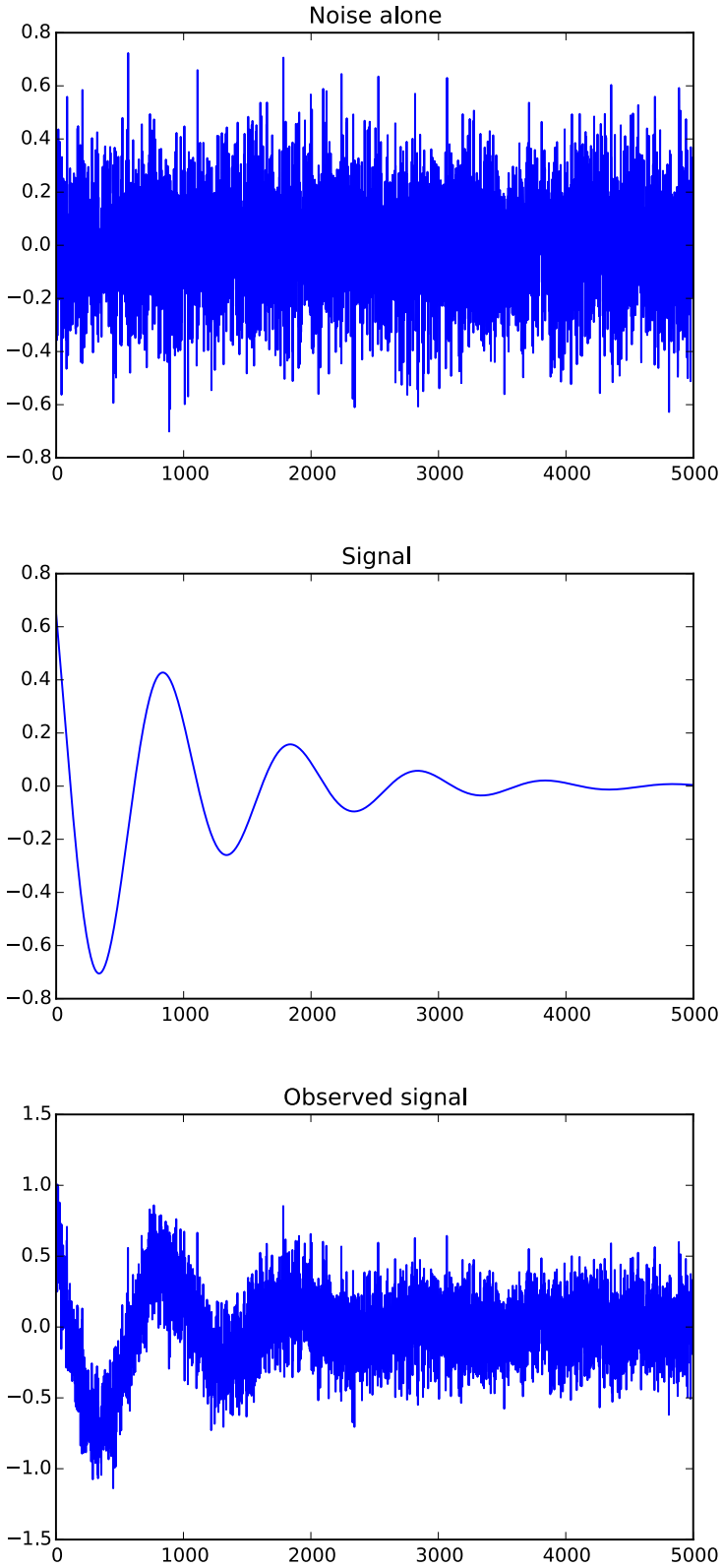
```
A=0.2; N=5000      t=arange(N)      s=exp(-0.001*t)*sin(2*pi*0.001*t+2*pi*rand(1))
w=A*randn(N)      x=s+w
```

It can be shown that the optimum Wiener filter is such that

$$H(f) = \frac{S_{ss}(f)}{S_{SS}(f) + S_{VV}(f)}, \tag{1.99}$$

where $S_{SS}(f)$ and $S_{VV}(f)$ are respectively the power spectra of the signal and of the noise. Implement this filter and compute its output. In practice, what must be known in order to implement this filter? Is this reasonable? Look at the impulse response and comment. What are the other difficulties for implementation?

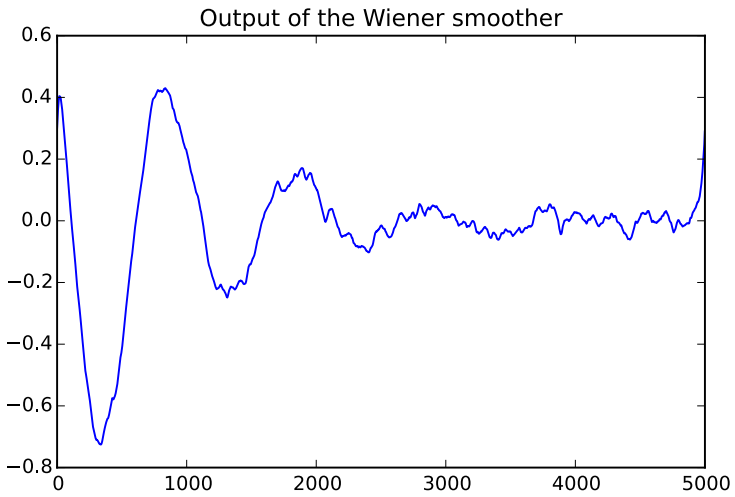
```
A=0.2
N=5000
t=arange(N)
s=exp(-0.001*t)*sin(2*pi*0.001*t+2*pi*rand(1))
w=A*randn(N)
figure(1); plot(w); title('Noise alone')
x=s+w
figure(2); plot(s); title('Signal')
figure(3); plot(x); title('Observed signal')
```



Implementation

```
Sss=1/N*abs(fft(s))**2
```

```
Svv=A*A*ones(N)
H=Sss/(Sss+Svv)
xx=real(ifft(H*fft(x)))

plot(xx)
title('Output of the Wiener smoother')
```



The drawbacks are that

- One must know the spectra of the signal and of the noise. Here we have suppposed that the noise is white and that we knew its variance. Furthermore, we assumed that the spectrum of the signal is known.
- The impulse response may have an infinite support and is not causal. For implementation in real time, on should select a causal solution. This requires to perform a spectral factorization and this is another story, see here or here, page 208 for details.

Wiener Smoother in the time domain

We now look for the expression of an optimal smoother in the time domain. Of course, we could simply take the impulse response associated with the frequency response (1.97). However, as we saw above, this impulse response is non-causal and has infinite duration. Instead, we shall reformulate the problem to include the fact that we look for a causal finite impulse response. We begin with the observation equation

$$x(n) = s(n) + v(n). \tag{1.100}$$

and we look for the filter with impulse response $w(n)$ such that $y(n) = [w * x](n)$ is as near as possible of $s(n)$: this can be formulated as the search for w which minimizes the mean square error

$$\mathbb{E}\left[([w * x](n) - s(n))^2\right]. \tag{1.101}$$

For a FIR filter, the convolution cans be written as the scalar product $ $

$$y(n) = [w * x](n) = \sum_{m=0}^{p-1} w(m)x(n-m) = \mathbf{w}^t\mathbf{x}(n) \tag{1.102}$$

where $\mathbf{w}^t = [w(0, w(1) \dots w(p-1)]$ and $\mathbf{x}(n)^t = [x(n), x(n-1), \dots x(n-p+1)]$. The mean square error can then be written as the function of $\mathbf{w}$

$$J(\mathbf{w}) = \mathbb{E}\left[\left(\mathbf{w}^t\mathbf{x}(n) - s(n)\right)^2\right]. \tag{1.103}$$

By the chain rule for differentiation and the fact that

$$\frac{\mathrm{d}\mathbf{w}^t\mathbf{x}(n)}{\mathrm{d}\mathbf{w}} = \mathbf{x}(n), \tag{1.104}$$

we get that

$$\begin{aligned}
\frac{\mathrm{d}J(\mathbf{w})}{\mathrm{d}\mathbf{w}} &= 2\mathbb{E}\left[\mathbf{x}(n)\left(\mathbf{w}^t\mathbf{x}(n) - s(n)\right)\right] && (1.105)\\
&= 2\mathbb{E}\left[\mathbf{x}(n)\left(\mathbf{x}^t\mathbf{w}(n) - s(n)\right)\right], && (1.106)\\
&= 2\mathbb{E}\left[\mathbf{x}(n)\mathbf{x}(n)^t\right]\mathbf{w} - \mathbb{E}\left[\mathbf{x}(n)s(n)\right]. && (1.107)\\
& && (1.108)
\end{aligned}$$

The first term involves a correlation matrix of $\mathbf{x}(n)$ and the second the vector of cross correlations between $\mathbf{x}(n)$ and $s(n)$. Denoting

$$\begin{cases} \mathbf{R}_{XX} = \mathbb{E}\left[\mathbf{x}(n)\mathbf{x}(n)^t\right], \\ \mathbf{r}_{SX} = \mathbb{E}\left[\mathbf{x}(n)s(n)\right] \end{cases} \tag{1.109}$$

we obtain

$$\mathbf{R}_{XX}\mathbf{w} = \mathbf{r}_{SX} \tag{1.110}$$

or

$$\boxed{\mathbf{w} = \mathbf{R}_{XX}^{-1}\mathbf{r}_{SX}} \tag{1.111}$$

if $\mathbf{R}_{XX}$ is invertible.