# MAP555 - Signal processing- Practical Session 2
# Random Signal Processing

The practical session will be done in Python 3 and it is strongly recommended to have a working Anaconda environnement. The different sections of the session should be implemented in Jupyter notebooks and it is strongly recommended to take notes in the notebook during the session.

The individual report for the session will be uploaded on moodle in python notebook format. It is expected to have a working code (reproducible figures) and a short discussion in markdown format for all the results obtained in the practicals session. The end of the report must contain a personal discussion about the session (what was hard to understand and implement, how you would do it newt time, what was new, discussion of relation with the course, personal discussion about how to use theses tools in a professional setting, ...).

#### Importing libraries

In this section we will use Numpy/Scipy Python libraries for handling numerical data and Matplotlib for plotting them. We will also need to have access to some function in the `scipy.signal` and `scipy.io.wavfile` submodule that have to be imported also.

```
import numpy as np
import pylab as pl
import scipy as sp
import scipy.signal
import scipy.io.wavfile
```

## 1 Random signals properties

### 1.1 Gaussian noise

1. Generate `w` a Gaussian IID random signal of $N_s = 1024$ samples and variance $\sigma^2 = 1.2$ (`np.random.randn`).

2. Plot the signal and check that its variance is around its theoretical value (`np.std,pl.plot`).

3. Plot an histogram of the signal, does it look like a gaussian? look what happens for different realization of the signal and for different number of samples $N_s$ (`pl.hist`).

4. Compute the empirical autocorrelation using the option `mode='full'` that computes the empirical correlation with zero-padding (`sp.signal.correlate`).

5. Plot simultaneously with the correct x-axis the empirical autocorrelation and its theoretical counterpart. Do you recover the variance of the noise for $n = 0$? What is happening at the extremities of the correlation?

6. Compute the empirical Power Spectral density (PSD) of the signal from the autocorrelation with mode `mode='same'` in order to have a FT of the same length of the signal (`np.fft.fft`). Is `Sw` a positive real? If not, why?

7. Compute another estimation of the PSD of `w` as the squared magnitude of the FFT of the realization `w`. You can do this because and $R_w[n] = W[n] \star W[-n]$ and $S_w = \mathcal{F}[R_w[n]] = \mathcal{F}[w(n)]\mathcal{F}[w(-n)] = |\mathcal{F}[w(n)]|^2$.
8. Plot both estimations of the PSD on the same graph. Why are they different?
9. Run and plot the PSD estimation for different number of samples $N_s$. What is the effect of $N_s$.
10. Generate an IID Gaussian noise of 3 seconds at frequency $f_s = 8000KHz$ and save it as a wave file (`sp.io.wavfile.write`). Does it sound like the beginning of an episode of an HBO series?

### 1.2 Noisy sine wave

We now work with the following stochastic process

$$X[n] = \cos(2\pi f_0 n + \phi) + W[n]$$

where $W[n]$ is the gaussian process in the previous section, $f_0 = 0.1$ and $\phi \sim \mathcal{U}(0, 2\pi)$ is a uniform random variable.

1. Generate a realization of the random signal $X[n]$. Plot it and discuss if the sine wave is visible in the signal.
2. Compute the empirical autocorrelation and plot it. Can you recognize the different components of the signal ?
3. Express the theoretical autocorrelation and plot it simultaneously with the empirical autocorrelation (with proper scaling).
4. Compute the empirical Power Spectral density (PSD) of the signal as the squared magnitude of the FFT of the realization `x`.
5. Compute and plot the PSD. Is the component from the signal well distinguishable?
6. Generate a signal `x` of $N_s = 3 * 8000$ sample and save it as a wave file at frequency $f_s = 8000KHz$ (`sp.io.wavfile.write`). Listen and compare it to the IID noise. Can you hear the frequency of the cosine?

## 2 AR modeling

### 2.1 Simulated signals

In this section we will study two AR models. The first one can be modeled as

$$X_1[n] - 0.9X_1[n-1] = W[n], \qquad a_1 = [1, -0.9]$$

It is aan AR model of order $N = 1$ and the filter $a$ is of size 2 because the first is set to 1 to recover the whole filter.

The second AR model is of order $N = 2$ and can be expressed as:

$$X_2[n] - 0.9X_2[n-1] + 0.8X[n-2] = W[n], \qquad a_2 = [1, -0.9, 0.8]$$

1. Generate a random signal `w` of $N_s = 4096$ samples (`np.random.randn`). Use this signal to generate both `x1` and `x2` corresponding to model 1 and 2 above (`sp.signal.lfilter`).
2. Plot the Gaussian process and both signals in time and their FFT. What kind of filtering is done by the two systems?

3. Generate 1 second of each of the signals and 8000Hz sampling and save them in wave files. Listen to them.

4. Code the estimation of the AR model using the following steps :

   - Compute the autocorrelation `r` of the signal (`sp.signal.correlate`).
   - Extract from `r` the first column `c` of the matrix $\mathbf{R}_X$ of the Yule-Walker equation and the vector $\mathbf{r}_X$ of the equations.
   - Solve the linear system for a Toeplitz matrix to estimate the AR coefficients with the Levinson recursion (`sp.linalg.solve_toeplitz`) .
   - Create the final estimated vector `ae` with a 1 as its first component (`np.concatenate`).

5. Create a function `def ar_yule(x,N):` that performs all the operations above and returns the estimated AR vector `ae`.

6. Estimate the AR parameters for both signals. Are the estimated value close to the true model?

7. What is the effect of changing number of samples $N_s$? What happens when the order $N$ of the AR model is larger than the true order of the model?

8. Code a function `get_psd_ar(a,f):` that returns the theoretical PSD of the AR process at frequencies in the array `f`.

9. Plot the PSD of the exact model and the estimated model for both signals. How close are they ? Do you recover the intuition you had from the FFT visualization about the kind filters ? What happens for different values of $N_s$?

10. Save 1 second of each AR signals for each model and listen to them. Which one contains the most high frequencies?

## 2.2 AR modeling of real life signals

1. Load the signal `uku.wav` that contains a note played on a ukulele (`sp.io.wavfile.read`). Plot its spectrum (`np.fft.fft`). What is the fundamental frequency of the note played (both in real Hz and in normalized frequency)? Store the frequency in variable `f0`.

2. Estimate an AR model for the signal of order $N = 100$ using the function you implemented earlier. Plot the estimated PSD. Is it similar to the spectrum of the signal?

3. Generate a 3 second signal using the AR model with a Gaussian noise excitation signal (`sp.signal.lfilter`). Save it as a wave file and listen to it. Conclusion?

4. Generate a 3 second signal using a dirac comb excitation signal at the fundamental frequency estimated earlier. Save it as a wave file and listen to it. Which excitation fits better to the current signal?

5. Do the previous steps again for the signal `uku2.wav`.

6. Bonus question: estimate AR coefficients for `conso.npz` for linear prediction (equivalent estimation). Does it provide a precise prediction as a FIR filter (you have to use the estimated AR coefficient with a minus sign and 0 as the first component of the filter)?

# 3 Wiener filtering

## 3.1 Simulated signal

1. Generate the noisy sine wave of section 1.2 for $N = 1024$ samples and store the noiseless signal `y` in memory.
2. Compute the theoretical autocorrelation and express the Wiener linear problem with Toeplitz matrix and solver (`sp.linalg.solve_toeplitz`). Estimate the Optimal filter for the current random process with $N = 10$.
3. Apply the filter and compare the filtered signal to the noiseless signal and original signal.
4. Compute the SNR for the noisy and filtered signal for different values of $N$. What is the optimal value?

## 3.2 Filtering noise in real life signal

1. Load the signal `y` from file `stairway.wav`. Listen to it and visualize its spectrum.
2. Create the signal `x` that contains `y` with some additive gaussian noise of same variance as the signal (SNR=0dB). Save the signal as `stairwayb2.wav` and listen to it.
3. Implement a causal wiener filter $h$ for the signal using the Wiener-Hopf equations and empirical estimates of $R_X$ and $R_{XY} = R_Y$ (the second one using the clean signal `y`).
4. Compute the filtered signal and save it to a wave file. Compute the SNR before and after filtering. What is the effect of the size $N$ of the filter? For which value of $N$ the SNR reaches a plateau what is the corresponding delay for real time applications.
5. Estimate the Wiener filter when you only have access to a proxy clean signal `y2` instead of the clean signal `y`. The proxy signal is available in `stairway2.wav`. Is the reconstruction as good in terms of audio quality and SNR?
6. When the signal is processed offline, one does not need to use a causal filter and the Wiener filtering can be done in the Fourier domain with FFT. Compute it using the theoretical PSD of the IID noise (see course) and the clean signal `y` to estimate $S_Y$. Listen to the filtering and compute the SNR.
7. Compute the FFT Wiener filtering when using the proxy `y2` instead of the clean signal for estimating $S_Y$. Comments on all the variants of Wiener filter