Lab random signal processing

S1: Autocorrelation estimators

Objectives of the first lab session: discovery of the topic, reliable estimation of the autocorrelation, criteria to determine the voiced or unvoiced nature of a sound

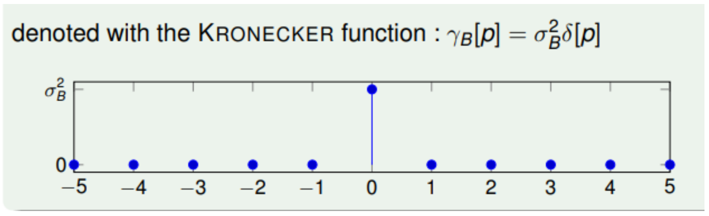
1) sample data

Here we created the random proceses we know the autocorrelation function

First we need some constants, to all the programs to work

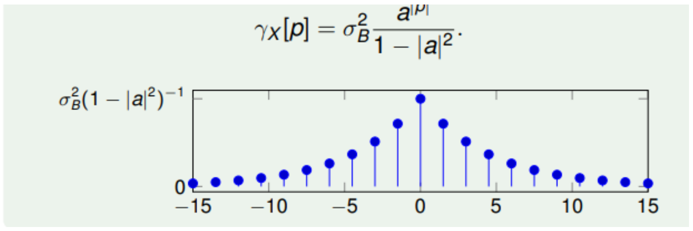
```
clearvars
close all
clc
%Number of samples of the noise
N = 500;

%Maximun number of p shifts
P_max = N-1;

%sampling frequency and period
Fs = 10000;
Ts = 1./Fs;

%Unvoiced file names
unvoiced_names = [...
    "data/nonvoise_ch.flac" "data/nonvoise_hw.flac" ...
    "data/nonvoise_kss.flac" "data/nonvoise_th.flac" ...
    ];

%Unvoiced audios and Ts
[unvoiced_audios, unvoiced_Ts] = audioReadChain(unvoiced_names,1024);

%voiced file names
voiced_names = [...
    "data/voise_nng.flac" "data/voise_oo.flac" ...
    "data/voise_ooo.flac" ...
    ];

%Voiced audios and Ts
[voiced_audios, voiced_Ts] = audioReadChain(voiced_names,1024);
```

White Noise (normaly distributed):


denoted with the KRONECKER function : $\gamma_B[p] = \sigma_B^2\delta[p]$

```
white_noise_variance = 1;
white_noise_deviation = sqrt(white_noise_variance);
white_noise_mean = 0;

white_noise = white_noise_mean + white_noise_deviation .*randn(1,N);

white_noise_autocorr = zeros(1,N);
white_noise_autocorr(1) = white_noise_variance;
```

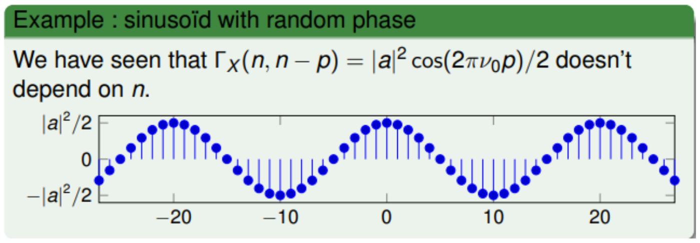
AR(1) Function:



$$\gamma x[p] = \sigma_B^2 \frac{a^{|p|}}{1-|a|^2}.$$

```
AR1_variance = 1;
AR1_deviation = sqrt(AR1_variance);
AR1_coefficient = 0.9;

AR1 = filter(1, [1 -AR1_coefficient],AR1_deviation.*randn(1,N));

AR1_autocorr = AR1_variance .* AR1_coefficient.^abs(0:P_max);
AR1_autocorr = AR1_autocorr./ (1-abs(AR1_coefficient).^2);
```

Sine wave with random phase (uniformaly distributed):



Example : sinusoïd with random phase

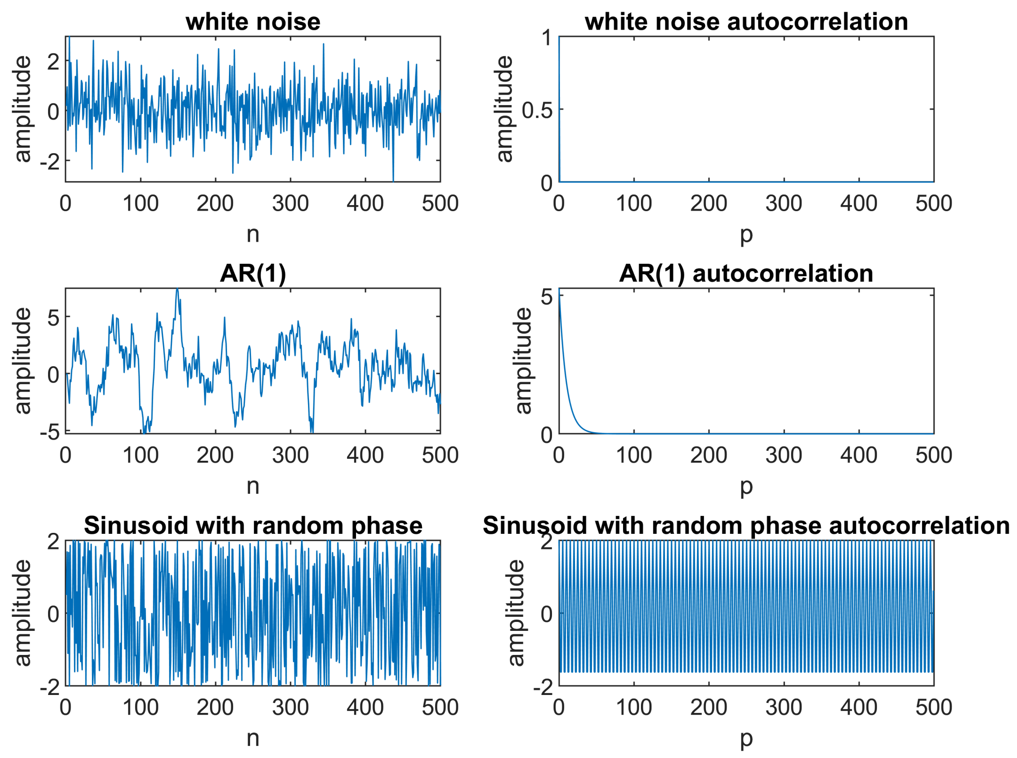
We have seen that $\Gamma_X(n, n-p) = |a|^2 \cos(2\pi\nu_0 p)/2$ doesn't depend on n.

```
sine_amplitude = 2;
sine_reduced_frequency = 0.20; % range [0,0.5)
sine_phase_shifts = 2.*pi.*rand(1,N);
sine_angules = 2.*pi.*sine_reduced_frequency.*(1:N);

sine = sine_amplitude.*sin(sine_angules + sine_phase_shifts);

sine_autocorr = (abs(sine_amplitude).^2) ./ 2;
sine_autocorr = sine_autocorr .* cos(2.*pi.*sine_reduced_frequency.*(0:P_max));
```

Now we graph them

```
%plot
figure(1)
x_axis_n = 1:N;
```

```
x_axis_p = 0:P_max;

subplot(3,2,1)
plot(x_axis_n,white_noise)
title("white noise")
xlabel("n"), ylabel("amplitude")

subplot(3,2,2)
plot(x_axis_p,white_noise_autocorr)
title("white noise autocorrelation")
xlabel("p"), ylabel("amplitude")

subplot(3,2,3)
plot(x_axis_n,AR1)
title("AR(1)")
xlabel("n"), ylabel("amplitude")

subplot(3,2,4)
plot(x_axis_p,AR1_autocorr)
title("AR(1) autocorrelation")
xlabel("p"), ylabel("amplitude")

subplot(3,2,5)
plot(x_axis_n,sine)
title("Sinusoid with random phase")
xlabel("n"), ylabel("amplitude")

subplot(3,2,6)
plot(x_axis_p,sine_autocorr)
title("Sinusoid with random phase autocorrelation")
xlabel("p"), ylabel("amplitude")
```



2) Biased autocorrelation estimator

we created a funcion (inside its own script) to implement the Biased autocorrelation estimator[1] .

Inputs:

- **X**: vector of signal samples
- **pmax**: maximal amount of shift to be considered (optional, defaults to the length of X minus 1)

Outputs:

- **Cx**: vector of cross correlation samples for shifts varying from 0 to pmax
- **p**: vector of corresponding shifts

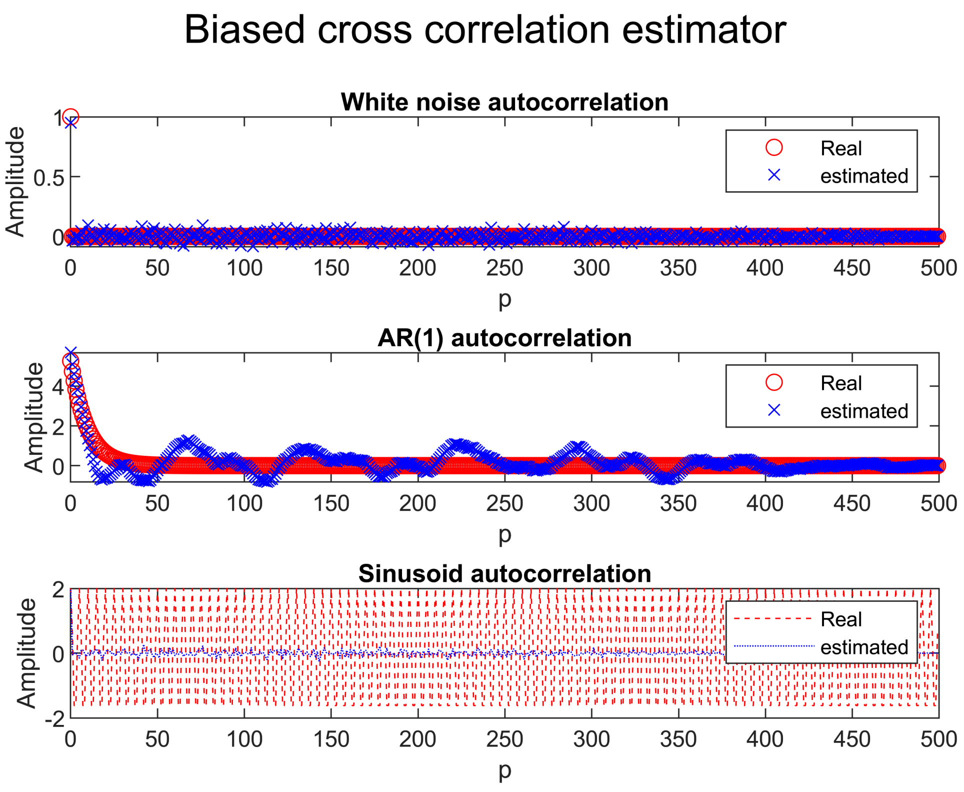
[1] [We decided to not change the name of the function, but we think that the name is not the correct one. The title of the section is biased autocorrelation estimator, but the function is caled biased cross correlation, and only accepts one vector of signal samples].

$$\widehat{\gamma_{\mathrm{Xb}}}(p,\omega)\frac{1}{N}\sum_{k=p}^{N-1}X(k,\omega)X(k,\omega)^*$$

testing

Now we test this stimator against the real autocorrelation of the functions:

```
%plot
figure(2)
sgtitle("Biased cross correlation estimator")
est_white_noise = BiasedCrossCorr(white_noise);
est_AR1 = BiasedCrossCorr(AR1);
est_sine = BiasedCrossCorr(sine);

subplot(3,1,1)
plot(x_axis_p,white_noise_autocorr,"ro")
hold on
plot(x_axis_p,est_white_noise,"bx")
title("White noise autocorrelation")
xlabel("p")
ylabel("Amplitude")
legend("Real","estimated")

subplot(3,1,2)
plot(x_axis_p,AR1_autocorr,"ro")
hold on
plot(x_axis_p,est_AR1,"bx")
title("AR(1) autocorrelation")
xlabel("p")
ylabel("Amplitude")
legend("Real","estimated")

subplot(3,1,3)
plot(x_axis_p,sine_autocorr,"r--")
hold on
plot(x_axis_p,est_sine,"b:")
title("Sinusoid autocorrelation")
xlabel("p")
ylabel("Amplitude")
legend("Real","estimated")
```

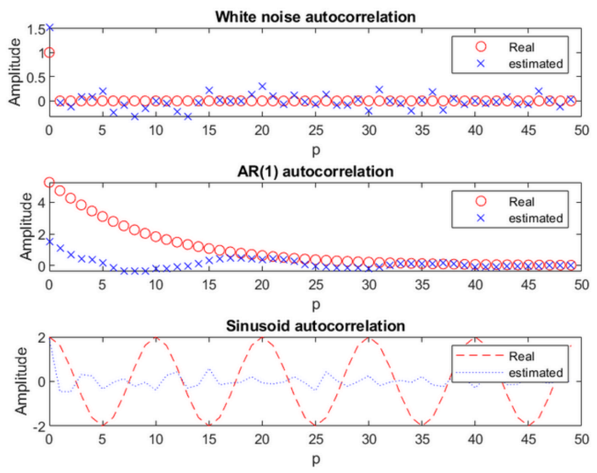


Then, were are the questions and the answers:

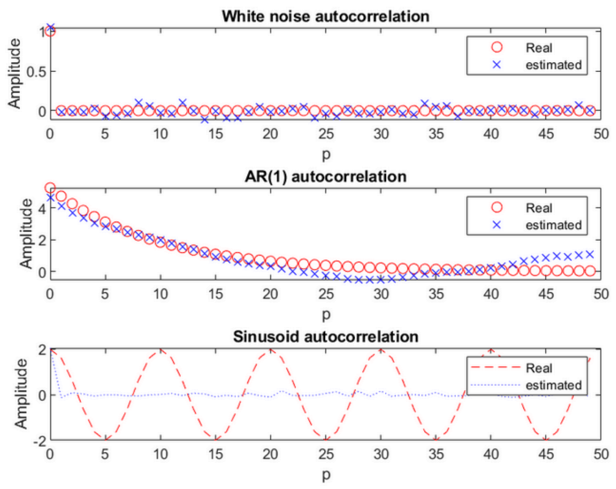
- For random processes for which you know the theoretical autocorrelation function, does the biased estimator converge when the number of samples grows?

Yes:

N=50



N=500 (First 50 values)



- Can you observe the effect of the bias through certain examples?

There seems to be some problems with the sinusoid, and the values of the AR(1) seems to be a bit lower in the middle of the graph

- Does the estimated autocorrelation for shift p = 0 match the output of the var function?

```
disp("estimated | variance")
```

```
estimated | variance
```

```
disp("--------------------")
```

```
--------------------
```

```
disp(est_white_noise(1) + "     | " + var(white_noise))
```

```
0.95089     | 0.95156
```

```
disp(est_AR1(1) + "     | " + var(AR1))
```

```
5.6821     | 5.3559
```

```
disp(est_sine(1) + "     | " + var(sine))
```

```
1.9566     | 1.9569
```

For our value of N (N=500 at the moment of answering), it seems that is almost identical.

- Is the biased estimator more suitable for voiced or unvoiced sounds?

we have this text from the guide:

- **voiced**, corresponding to vocal vibrations, modeled by a periodic pulse with random phase (this for example the case for vowels, and for the sounds ‘b’, ‘d’ and ‘z’),
- **unvoiced**, corresponding to a turbulent flow of exhaled air, modeled by a white noise (this is for example the case for the sounds ‘p’, ‘t’, ‘s’ and ‘ch’)

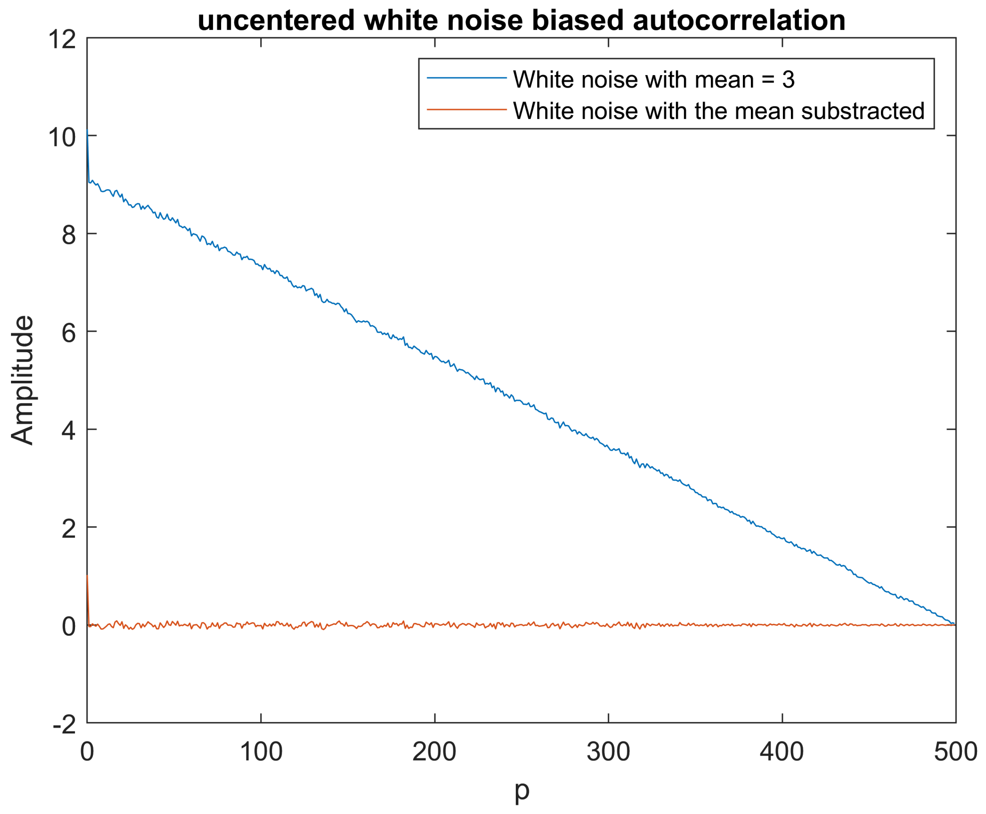
in this case, it is better for unvoiced sounds, as it does not work well with sinusoids and works really well with white noises.

- What happens if the signal contains a continuous component, representing a random process which is not zero-mean? How to correctly estimate its autocorrelation?

```
%Generation of a not centered whithe noise
white_noise_variance_unc = 1;
white_noise_deviation_unc = sqrt(white_noise_variance_unc);
white_noise_mean_unc = 3;

white_noise_unc = white_noise_mean_unc + white_noise_deviation_unc .*randn(1,N);

%plot
figure(3)
plot(x_axis_p,BiasedCrossCorr(white_noise_unc))
title("uncentered white noise biased autocorrelation")
xlabel("p")
ylabel("Amplitude")
```

We can see that the stimation became something really different. To correctly stimate the autocorrelation, we have to estimate the mean effect too. In this case, substracting the mean to the value before the autocorrelation works (code below)

```
%plot continuation
hold on
plot(x_axis_p,BiasedCrossCorr(white_noise_unc - mean(white_noise_unc)))
legend("White noise with mean = "+white_noise_mean_unc, "White noise with the mean substracted"
```



3) Unbiased autocorrelation estimator

we created a funcion (inside its own script) to implement the unbiased autocorrelation estimator[2].

Inputs:

- **X**: vector of signal samples
- **pmax**: maximal amount of shift to be considered (optional, defaults to the length of X minus 1)

Outputs:

- **Cx**: vector of cross correlation samples for shifts varying from 0 to pmax
- **p**: vector of corresponding shifts

[2][As the previous one, we decided to not change the name of the function, but we think that the name is not the correct one. The title of the section is unbiased autocorrelation estimator, but the function is caled unbiased cross correlation, and only accepts one vector of signal samples].

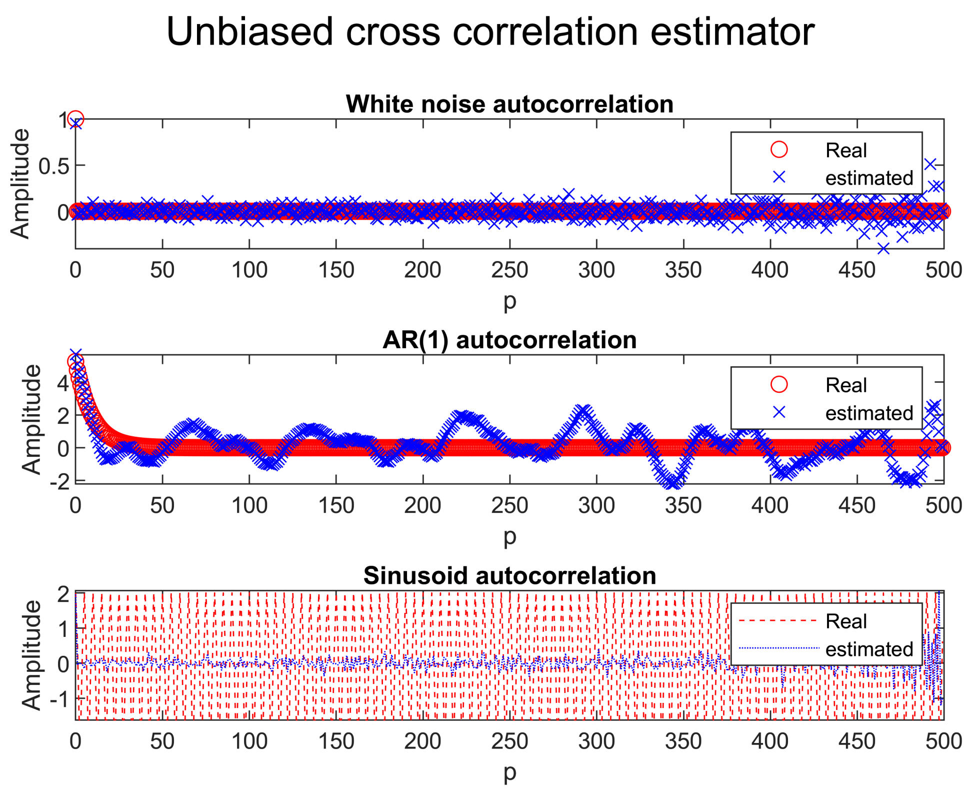
$$\widehat{\gamma_{\mathrm{Xnb}}}(p,\omega)\,\frac{1}{N-p}\sum_{k=p}^{N-1}X(k,\omega)X(k,\omega)^{*}$$

Test:

testing

Now we test this stimator against the real autocorrelation of the functions:

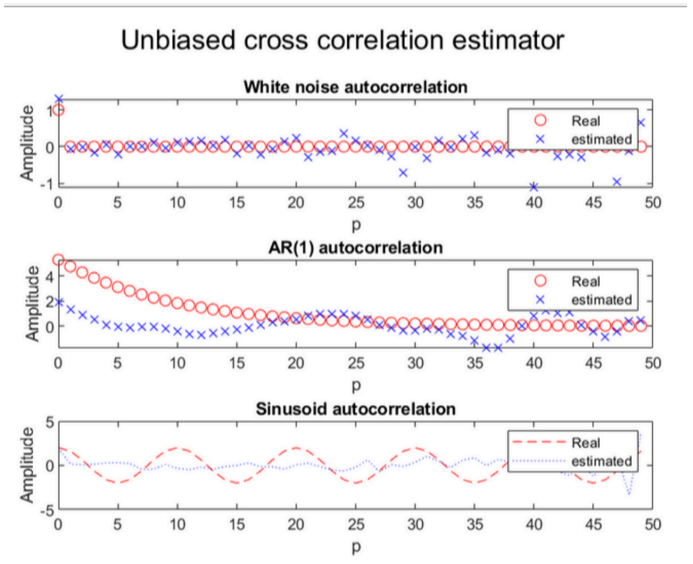
```
%plot
figure(4)
sgtitle("Unbiased cross correlation estimator")
est2_white_noise = UnbiasedCrossCorr(white_noise);
est2_AR1 = UnbiasedCrossCorr(AR1);
est2_sine = UnbiasedCrossCorr(sine);

subplot(3,1,1)
plot(x_axis_p,white_noise_autocorr,"ro")
hold on
plot(x_axis_p,est2_white_noise,"bx")
title("White noise autocorrelation")
xlabel("p")
ylabel("Amplitude")
legend("Real","estimated")

subplot(3,1,2)
plot(x_axis_p,AR1_autocorr,"ro")
hold on
plot(x_axis_p,est2_AR1,"bx")
title("AR(1) autocorrelation")
xlabel("p")
ylabel("Amplitude")
legend("Real","estimated")

subplot(3,1,3)
plot(x_axis_p,sine_autocorr,"r--")
hold on
plot(x_axis_p,est2_sine,"b:")
title("Sinusoid autocorrelation")
xlabel("p")
ylabel("Amplitude")
legend("Real","estimated")
```



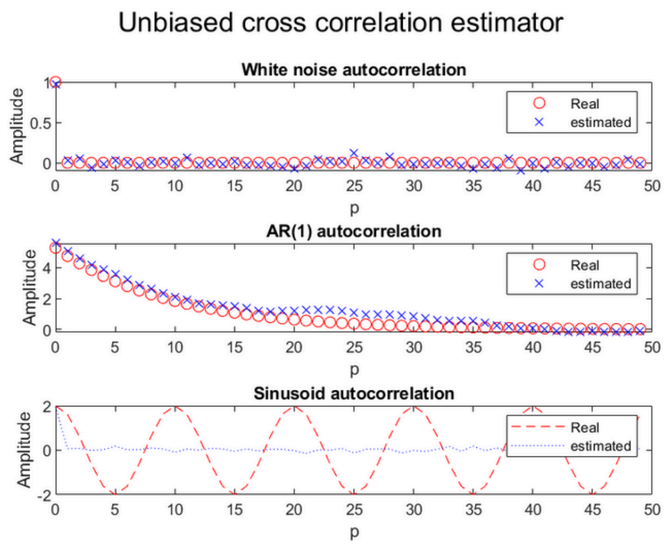
Then, were are the questions and the answers:

- For random processes for which the theoretical autocorrelation is known, does the unbiased estimator converge when the number of samples grows? Do you actually observe a zero bias?

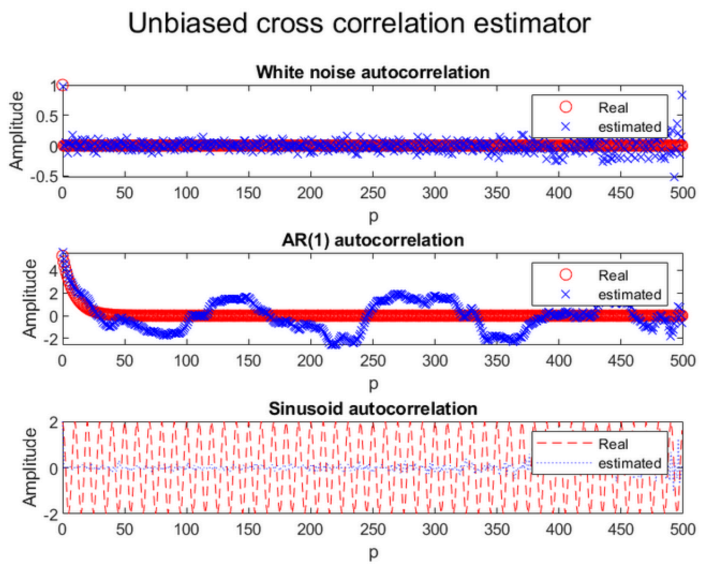
N=50



N=500 (first 50)



N=500



It seems to converge, but there is a problem of convergence at the last values of the shift. I do not observe a 0 bias, but it seems to have done something.

- Can you observe the effect of the variance of the estimator through certain examples?

```
disp("     Estimation        |    Real   ")
```

```
     Estimation        |    Real
```

```
disp("unbiased   | biased   | variance")
```

```
unbiased   | biased   | variance
```

```
disp("----------------------------")
```

```
----------------------------
```

```
disp(est2_white_noise(1) + "    | " + est_white_noise(1) + "    | " + var(white_noise))
```

```
0.95089   | 0.95089  | 0.95156
```

```
disp(est2_AR1(1) + "     |  " + est_AR1(1) + "   |  " + var(AR1))
```

```
5.6821    | 5.6821  | 5.3559
```

```
disp(est2_sine(1) + "     |  " + est_sine(1) + "   |  " + var(sine))
```

```
1.9566    | 1.9566  | 1.9569
```

For our value of N (N=500 at the moment of answering), it seems that they are the same as in the unbiased ones. If we reduce N to 50, they are the same again, but they differ a considerable amount for the AR(1). If we reduce N to 10, they are the same again, but the error became bigger.

- Is the unbiased estimator more suitable for voiced or unvoiced sounds?

We would said voiced sounds, as we could see if we zoom that the sinusodial now have some meaningful pattern instead of something almost flat.

4) Establishing a criterion to determine the voiced / unvoiced type of sounds

1. The voiced sounds issued from speech present a fundamental frequency which varies between 100 Hz (man) and 400 Hz (child). Find the range of pseudo-periods that characterize voiced sounds.

Here, doing a bit of basic algebra, we have a pseudoperiod of

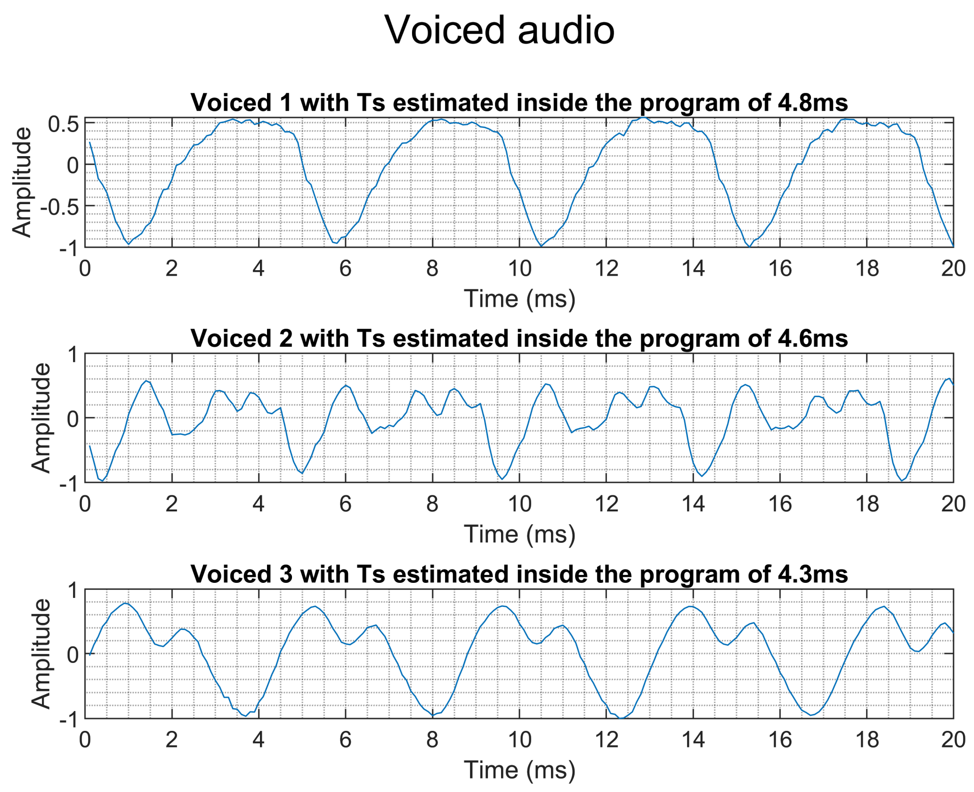
$$T_1 = \frac{1}{400\text{Hz}} = 2.5\text{ms} \;\; ; \; T_2 = \frac{1}{100\text{Hz}} = 10\text{ms}$$

2. Perform a temporal observation of voiced sounds to measure their pseudo-period.

Here we graph the voiced signals to see the pseudoperiod

```
%plot
figure(5)
number_of_samples = 200;
x_axis_t = Ts.*(1:1024).*1000;

subplot(3,1,1)
sgtitle("Voiced audio")

voiced1 = voiced_audios(1,:);
Min_prom = (max(voiced1) - min(voiced1))/2;
[~, peak_index_1] = findpeaks(voiced1,"MinPeakProminence",Min_prom);
est_Tsignal_1 = x_axis_t(peak_index_1(2)) - x_axis_t(peak_index_1(1));

plot(x_axis_t(1:number_of_samples), voiced1(1:number_of_samples))
title("Voiced 1 with Ts estimated inside the program of " + est_Tsignal_1 + "ms")
grid minor
xlabel("Time (ms)")
ylabel("Amplitude")

subplot(3,1,2)
```

```
voiced2 = voiced_audios(2,:);
Min_prom = (max(voiced2) - min(voiced2))/2;
[~, peak_index_2] = findpeaks(voiced2,"MinPeakProminence",Min_prom);
est_Tsignal_2 = x_axis_t(peak_index_2(2)) - x_axis_t(peak_index_2(1));

plot(x_axis_t(1:number_of_samples), voiced2(1:number_of_samples))
title("Voiced 2 with Ts estimated inside the program of " + est_Tsignal_2 + "ms")
grid minor
xlabel("Time (ms)")
ylabel("Amplitude")

subplot(3,1,3)

voiced3 = voiced_audios(3,:);
Min_prom = (max(voiced3) - min(voiced3))/2;
[~, peak_index_3] = findpeaks(voiced3,"MinPeakProminence",Min_prom);
est_Tsignal_3 = x_axis_t(peak_index_3(2)) - x_axis_t(peak_index_3(1));

plot(x_axis_t(1:number_of_samples), voiced3(1:number_of_samples))
title("Voiced 3 with Ts estimated inside the program of " + est_Tsignal_3 + "ms")
grid minor
xlabel("Time (ms)")
ylabel("Amplitude")
```



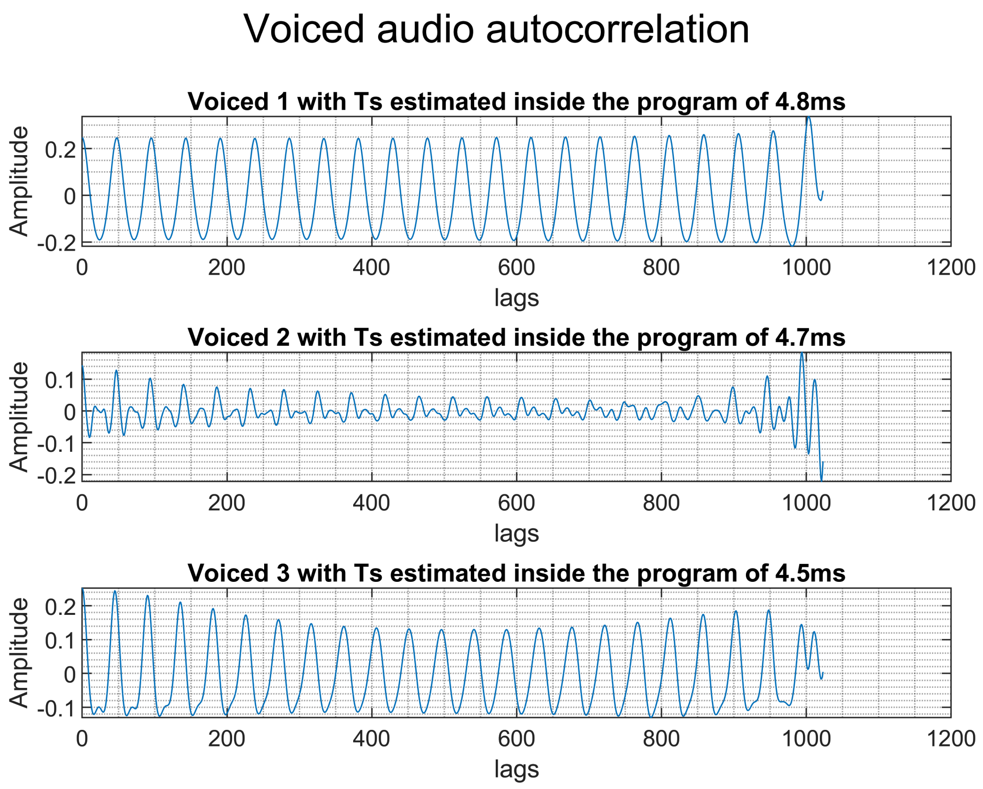
If we measure normaly, we estimate a pseudoperiod of 4.75ms, 4.6ms and 4.3ms respectively.

3. Check that you can recover the same pseudo-periods from the estimated autocorrelations.

Now, we do the same but with the autocorrelations

```
%plot
figure(6)
number_of_samples = 1024;
start_of_signal = 2;
x_axis_p = 0:1023;

subplot(3,1,1)
sgtitle("Voiced audio autocorrelation")

voiced1 = UnbiasedCrossCorr(voiced_audios(1,:));

Min_prominence = (max(voiced1) - min(voiced1))/2;
[~, peak_index_1] = findpeaks(voiced1(start_of_signal:number_of_samples),...
                              "MinPeakProminence",Min_prominence...
                              );
est_Tsignal_1 = (peak_index_1(1) + start_of_signal-2) .* Ts *1000;

plot(x_axis_p(1:number_of_samples), voiced1(1:number_of_samples))
title("Voiced 1 with Ts estimated inside the program of " + est_Tsignal_1 + "ms")
grid minor
xlabel("lags")
ylabel("Amplitude")

subplot(3,1,2)

voiced2 = UnbiasedCrossCorr(voiced_audios(2,:));

Min_prominence = (max(voiced2) - min(voiced2))/2;
[~, peak_index_2] = findpeaks(voiced2(start_of_signal:number_of_samples),...
                              "MinPeakProminence",Min_prominence...
                              );
est_Tsignal_2 = (peak_index_2(1) + start_of_signal-2) .* Ts *1000;

plot(x_axis_p(1:number_of_samples), voiced2(1:number_of_samples))
title("Voiced 2 with Ts estimated inside the program of " + est_Tsignal_2 + "ms")
grid minor
xlabel("lags")
ylabel("Amplitude")

subplot(3,1,3)

voiced3 = UnbiasedCrossCorr(voiced_audios(3,:));

Min_prominence = (max(voiced3) - min(voiced3))/2;
[~, peak_index_3] = findpeaks(voiced3(start_of_signal:number_of_samples),...
                              "MinPeakProminence",Min_prominence...
                              );
est_Tsignal_3 = (peak_index_3(1) + start_of_signal-2) .* Ts *1000;

plot(x_axis_p(1:number_of_samples), voiced3(1:number_of_samples))
```

```
title("Voiced 3 with Ts estimated inside the program of " + est_Tsignal_3 + "ms")
grid minor
xlabel("lags")
ylabel("Amplitude")
```



If we measure normally, we find a pseudoperiod of 4.8ms, 4.7ms, and 4.5ms respectively. (This was measured wtih the plot, not with the program, we especify)

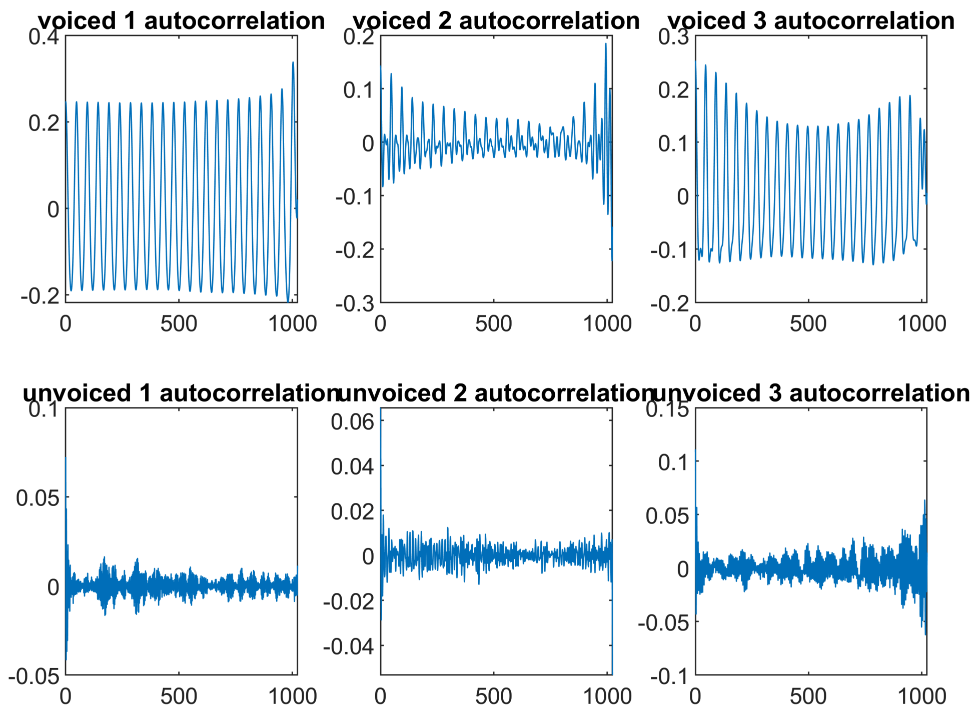
4. Observe how the “structure” of unvoiced signals makes the autocorrelation decrease when the shift increases. 5. Propose a criterion to distinguish voiced sounds from unvoiced sounds and implement it

First we plot the signals to see the phenomena

```
figure(7)
subplot(2,3,1)
sgtitle("Observation of the autocorrelation for 3 voiced and unvoiced sounds")
Y = UnbiasedCrossCorr(voiced_audios(1,:));
plot(x_axis_p,Y);
title("voiced 1 autocorrelation")

subplot(2,3,2)
Y = UnbiasedCrossCorr(voiced_audios(2,:));
plot(x_axis_p,Y);
title("voiced 2 autocorrelation")

subplot(2,3,3)
Y = UnbiasedCrossCorr(voiced_audios(3,:));
plot(x_axis_p,Y);
```

```
title("voiced 3 autocorrelation")

subplot(2,3,4)
Y = UnbiasedCrossCorr(unvoiced_audios(1,:));
plot(x_axis_p,Y);
title("unvoiced 1 autocorrelation")

subplot(2,3,5)
Y = UnbiasedCrossCorr(unvoiced_audios(2,:));
plot(x_axis_p,Y);
title("unvoiced 2 autocorrelation")

subplot(2,3,6)
Y = UnbiasedCrossCorr(unvoiced_audios(3,:));
plot(x_axis_p,Y);
title("unvoiced 3 autocorrelation")
```

ɔservation of the autocorrelation for 3 voiced and unvoiced soun



After looking at them, voiced sounds have some sort of big values compared to the unvoiced ones in the midle, and the values of unvoiced ones are really smal when they are not in the lower p's. One way we could detect if iis voiced or not could be to use a threshold that depends on the variance and the values of the peaks. Here we created a function for that:

Inputs:

- **`Cx`**: vector of cross correlation samples for shifts varying from 0 to numel(Cx)-1.

Outputs:

- bool: 1 (true) if the sound is voiced, 0 (false) otherwise.

Then we test it

```
%there could be plots if it gets uncommented in the function
%so we will be doing the count of the figures like it plotted
disp("Type      | predicted ")
```

```
Type      | predicted
```

```
disp("---------------------")
```

```
---------------------
```

```
for i=1:3
    disp("voiced    | "+ isVoiced(UnbiasedCrossCorr(voiced_audios(i,:))))
end
```

```
voiced    | true
voiced    | true
voiced    | true
```

```
for i=1:4
    disp("unvoiced | "+ isVoiced(UnbiasedCrossCorr(unvoiced_audios(i,:))))
end
```

```
unvoiced | false
unvoiced | false
unvoiced | false
unvoiced | false
```

S2 Spectral analysis

1) Sample data

As in the previous session, you can begin by testing your estimators using realizations of random processes for which you know the spectral features: white noise, AR1, and especially random phase sine waves. After these tests, you can go further and analyse the spectral features of voiced and unvoiced sounds in the repository provided.

We already tested our stimators, but it is not good on sinusoids as expected.

2) Two approaches for (almost . . .) the same estimator

In order to estimate the power spectral density (PSD) of a random process X whose trajectory has been observed over a finite number N of samples, there are two possible approaches:

- -correlogram: use the biased autocorrelation estimator and compute its Fourier transform,

- periodogram: compute the discrete-time Fourier transform (DTFT) of the available samples, take its squared modulus and divide by the number of samples.

In practice, it is not possible to estimate the power spectral density for all possible values of reduced frequency (the variable is continuous). We will thus compute the PSD over a finite set of reduced frequencies. You can exploit the FFT algorithm to compute discrete Fourier transforms, with a suitable choice of the parameter N_{fft}.

Write a function which implements the estimation of the PSD as described below:

Inputs:

- X: vector of signal samples
- Nfft: Number of signal samples to be considered in the discrete Fourier transform (see FFT) (optional argument, defaults to twice the length of X)

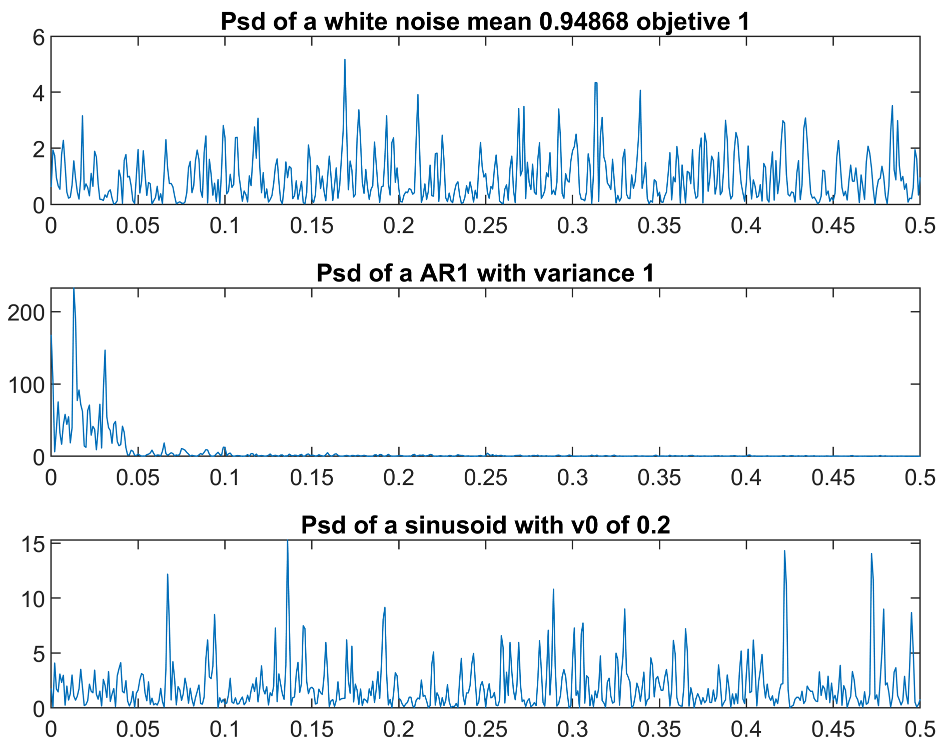
Outputs:

- PSD: vector of PSD estimates for normalized frequency varying from 0 to 1/2.
- nu: vector of corresponding normalized frequency values.

We created the funcion in another script.

Then we tested it with our signals

```
figure(8)
Nfft = 1000;

subplot(3,1,1)
[y,x] = psdEstimator(white_noise,Nfft);
plot(x,y)
title("Psd of a white noise mean "+ mean(y) + " objetive " + white_noise_variance)

subplot(3,1,2)
[y,x] = psdEstimator(AR1,Nfft);
plot(x,y)
title("Psd of a AR1 with variance " + AR1_variance)

subplot(3,1,3)
[y,x] = psdEstimator(sine,Nfft);
plot(x,y)
title("Psd of a sinusoid with v0 of "+ sine_reduced_frequency)
```



then, we answer the questions:

- What is the minimum value of Nfft such that the values returned by the FFT algorithm correspond to the expected discrete Fourier transform?

Nfft have to be two times the length of the signal.

- For the correlogram method, which values should you take for the shift?

for all the shifts from 0 to the length of the signal segment we want to analize

- For the correlogram method, can you write the vector of the autocorrelation coeficients in such a way that the coeficients computed using the FFT are purely real (and positive)?

[To do later]

- For the periodogram method, which number of samples should you use to renormalize the results? N or Nfft?

Nfft, as we know work with the frequency.

- What are the reduced frequencies corresponding to the coeficients returned by the FFT algorithm?

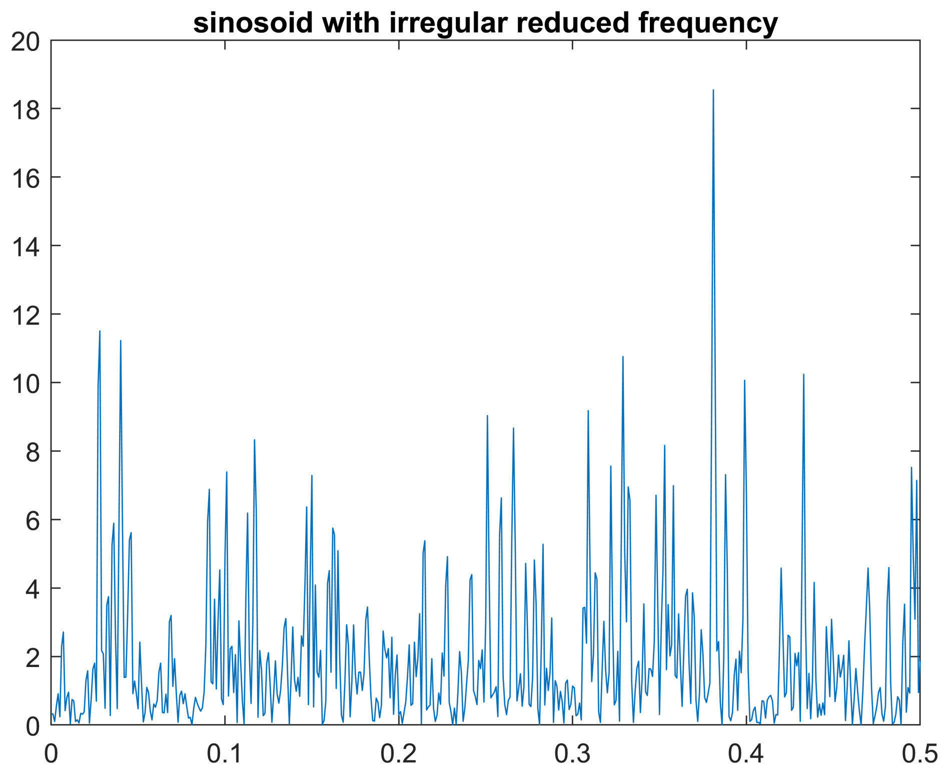
it returns the reduced frequencies form 0 to 1

- How to get only the spectral samples corresponding to a reduced frequency between 0 and 1/2?

Cutting the output samples so it correspond to the ones from 0 to 0.5

- Test your function with the realizations of a sine wave with random phase whose frequency v0 is an irrational number (for example v0 = √ 2/8).

```
sine_reduced_frequency_irr = sqrt(2)/8;
sine_angules_irr = 2.*pi.*sine_reduced_frequency_irr.*(1:N);

sine_irr = sine_amplitude.*sin(sine_angules_irr + sine_phase_shifts);
figure(15)
[y,x] = psdEstimator(sine_irr,Nfft);
plot(x,y)
title("sinosoid with irregular reduced frequency")
```

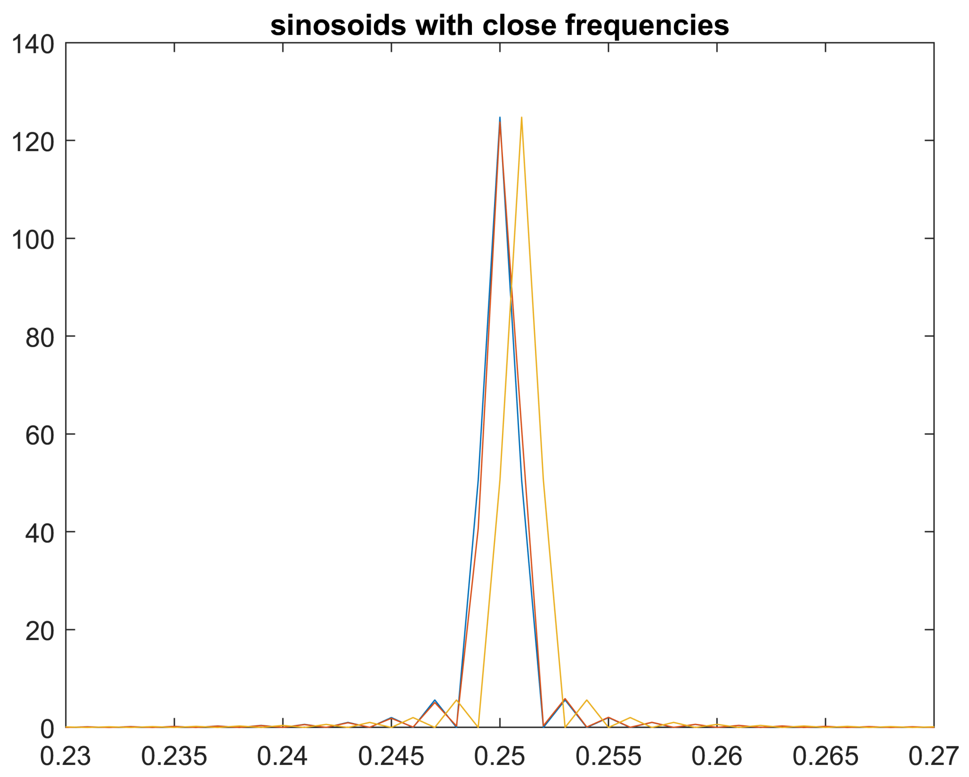


Here we see that we do not have the specific value (With the constans we selected at the moment), but we have values near that.

- Test the resolution capability of your estimator for the superposition of two sine waves whose frequences v1 and v2 are very close.

We have a resolution of 0.001 aproximately (With the constans we selected at the moment)

```
figure(16)
[y1,x1] = psdEstimator(sin(2.*pi.*0.25.*(1:N)),Nfft);
[y2,x2] = psdEstimator(sin(2.*pi.*0.2501.*(1:N)),Nfft);
[y3,x3] = psdEstimator(sin(2.*pi.*0.251.*(1:N)),Nfft);

plot(x1,y1,x2,y2,x3,y3)
title("sinosoids with close frequencies")
xlim([0.23 0.27])
```



Here we can see that the peak is in the same place for both, althought it is sightly different and one that has a change that can be seen with our resolution. (selected artifically)