University of Brasília

Electrical Engineering Department

Topics in Biomedical Engineering

Task 4

Authors:

Caio Luiz Candeias Flôres 190134283

Felipe Carneiro da Motta 200017616

João Pedro Daher Aranha 190109742

Brasília

September 2, 2022

Contents

1 Questão 1

Rode o caso 1D: função senoidal. Observe os gráficos obtidos. Veja que estão sendo plotados os gráficos das áreas de $|x^2(n)| = \text{abs}(x^2(n))$ e de $|X(k)|^2 = \text{abs}(|X^2(k)|)$, sendo as áreas calculadas como:

$$\frac{1}{N}\sum_{n=0}^{N-1} x^2(n) = \frac{1}{N^2}\sum_{n=0}^{N-1} |X(k)|^2 \Rightarrow \sum_{n=0}^{N-1} x^2(n) = \frac{1}{N}\sum_{n=0}^{N-1} |X(k)|^2, \;\; (A7.1-11)$$

1.1 1a.

Observe que, de fato, a equação (A7.1-11) está sendo satisfeita: compare os valores das variáveis $E1\_timedomain = sum(abs(x.^2))$ e $E1\_frequdomain = sum(abs(x.^2)/N$. No entanto, observe que, no gráfico de $fx = fft(x)$, não foi definido o eixo das frequências no gráfico. Por isso, o eixo horizontal do gráfico de fx não está na unidade de frequência. Mas, como o autor estava interessado na equação A7.1-11, esta equação foi implementada corretamente.

The MATLAB's code of Parseval-theorem.m:

```matlab
close all;
% 1D case :  Sinusoidal function
Fs=40; f=4; Ts=1/Fs;T=2; t=0:Ts:T-Ts;N=length(t);
x=2*cos(2*pi*f*t);
fx=fft(x);
figure,
subplot(1,2,1), area(t,abs(x.^2)),title(' Time Domain')
    ;
subplot(1,2,2),area(abs(fx)), title(' Frequency Domain'
    );
E1_timedomain=sum(abs(x.^2))
E1_frequdomain=sum(abs(fx.^2))/N
saveas(gcf, sprintf('%s.png', mfilename)); % save image
```

In this code, the Parseval Theorem, in the Equation 1, is being satisfied. The variables E1-timedomain and E1-frequdomain are equal to 160. Evidently, the energy of an periodic signal in the time domain is equal to energy of the transformed signal in the frequency domain divided by the amount of samples. However, the frequency axis wasn't defined and, in this code, this axis isn't the focus of observation.

$$\sum_{n=0}^{N-1} x^2(n) = \frac{1}{N}\sum_{k=0}^{N-1} |X(k)|^2 \tag{1}$$

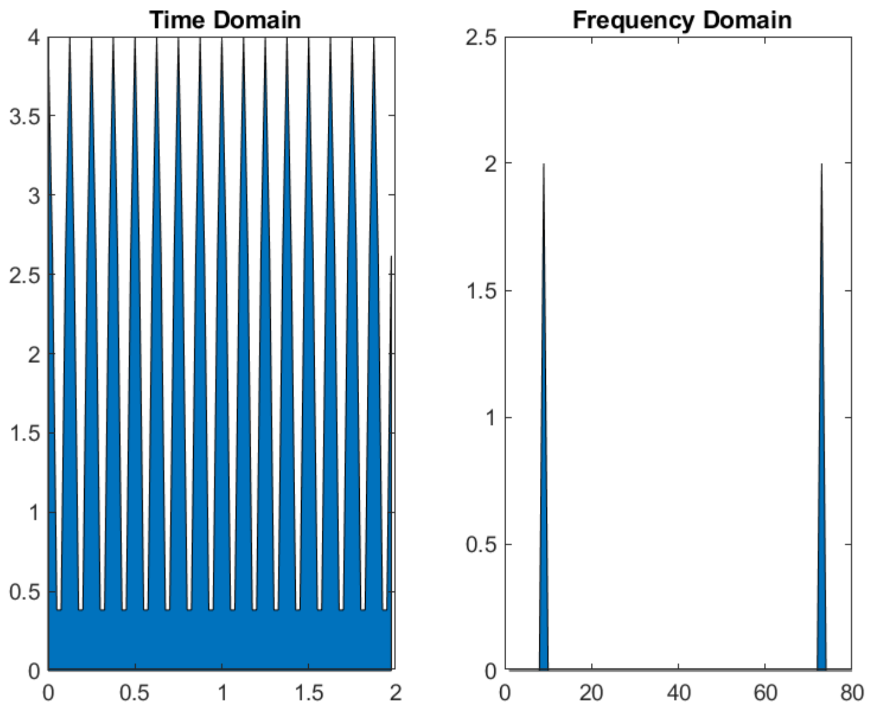


Figure 1: Graphic of the 1a

1.2 1b.

- Link de suporte.

Modifique o código do caso 1D (salvando o arquivo como "Parseval-theorem-modificado.m", por exemplo) para que a função plotada em suplot(1,2,2) tenha as unidades corretas no eixo da frequência e que mostre o gráfico de $|X{\cdot}X^*|/N = abs((X{\cdot}conj(X))/N)$, para melhor visualizar as áreas de fato sendo calculadas nas variáveis **E1_timedomain** e **E1_frequdomain**.

The MATLAB's code of Parseval-theorem-modificado.m:

```
close all;
% 1D case :  Sinusoidal function
Fs=40; % sampling frequency
f=4; % sinusoid frequency
Ts=1/Fs; % samoling period
T=2; % specific value of time
t=0:Ts:T-Ts; % time axis
N=length(t); % number of points
x=2*cos(2*pi*f*t);

f = (0:N-1)*Fs/N; % frequency axis
fx=(2/N)*fft(x); % fourier transform of x
fs_mag=abs(fx); % fftmagnitude

figure,
subplot(1,2,1), area(t,abs(x.^2)),title(' Time Domain')
    ;
subplot(1,2,2),area(f(1:N/2),fs_mag(1:N/2)), title('
    Frequency Domain'); % plot only until fs/2

% method 1
E1_timedomain = sum(abs(x.^2))
E1_frequdomain = sum(abs(fx.^2))/N

% method 2
E2_timedomain = sum(abs(x.^2))/N
E2_frequdomain = sum(abs(fx.^2))/N^2
saveas(gcf, sprintf('%s.png', mfilename)); % save image


disp('methods')
out = sprintf('E1_timed: %.4f, E1_freqd: %.4f',
    E1_timedomain, E1_frequdomain);
disp(out)
out = sprintf('E2_timed: %.4f, E2_freqd: %.4f',
    E2_timedomain, E2_frequdomain);
disp(out)
```

In this item, the original code was modified and an frequency axis was created with the intention to plot signal energy in the time domain and in the frequency domain with the correct units. In this plot, the frequency axis was restricted only to the positive frequencies $(\frac{fs}{2})$ in order to generate a better visualization and the magnitude of the energy was also normalized.

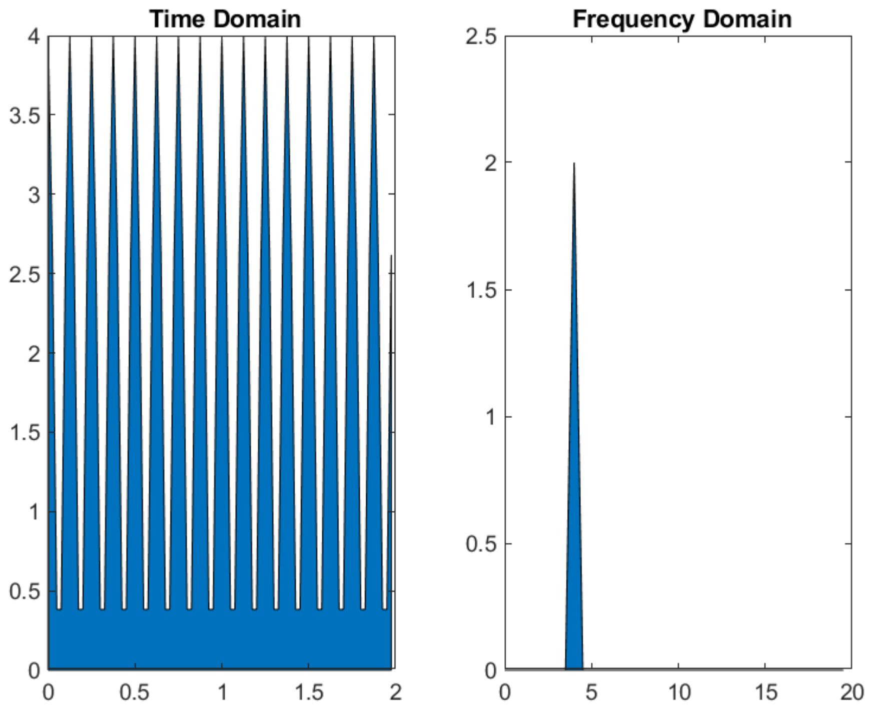
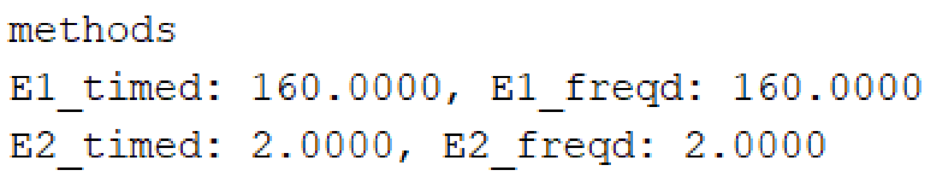


Figure 2: Graphic of the 1b.

It's possible to visualize, in the Figure 3, that the Parseval Theorem continues to be satisfied. The equations 2 and 3, which are two versions of the Parseval Theroem, are both respected.

$$\sum_{n=0}^{N-1} x^2(n) = \frac{1}{N} \sum_{k=0}^{N-1} |X(k)|^2 \tag{2}$$

$$\frac{1}{N} \sum_{n=0}^{N-1} x^2(n) = \frac{1}{N^2} \sum_{k=0}^{N-1} |X(k)|^2 \tag{3}$$



```
methods
E1_timed: 160.0000, E1_freqd: 160.0000
E2_timed: 2.0000, E2_freqd: 2.0000
```

Figure 3: Two versions of the Parseval Theorem.

2 Questão 2

Baseado nas funções da dissertação de Maria Stavriou (disponível em Link), verifique como a autora implementou o espectro de potência do sinal $f(x) = sin 2\pi 0.02t + sin 2\pi 0.1t + sin 2\pi 0.3t$, interpolado na frequência de 4 Hz, pelos métodos: (1) baseado em Fourier; (2) baseado no periodograma de Welch e (3) pelométodo paramétrico AR (item 5.1 do livro texto), implementado pelo algoritmo de Burg e pelo algoritmo de Yule.

- Os códigos estão disponíveis no site do curso, no arquivo zipado Link. O programa principal é o "TesteCodigoMaria_Parseval.m".
- Observe que a energia total do espectro de potência (no domínio da frequência) calculada por qualquer um destes métodos da Equação (A7.1-11) preserva a energia do sinal orignal no domínio do tempo.

First of all, the code used by Maria Stavriou, in "TesteCodigoMaria_Parseval.m", is the following:

```
Fs = 4;
t=0:1/Fs:(200-(1/Fs));
freqRes=Fs/length(t);
F1=0.07;
F2=0.3;
F3=0.1;
x=sin(2*pi.*t*F1)+cos(2*pi.*t*F2)+sin(2*pi.*t*F1);
figure; plot(t,x); hold on;
sprintf('A energia do sinal é dada por: %0.5g',sum(x
    .^2)/length(x))
%% Before interpolation
[power_timebefore_dsp powerfreqbeforedsp]=dspmaria_fft(
    x,Fs);
```

```
sprintf('A energia por fft é dada por: %0.5g',
    powerfreqbeforedsp)

[power_timebefore_W powerfreqbeforeW]=dspmaria_welch2(x
    ,Fs,64);
sprintf('A energia por Welch é dada por: %0.5g',
    powerfreqbeforedsp)

[power_timebefore_burg powerfreqbeforeburg]=
    dspmaria_burg(x,Fs,9);
sprintf('A energia por Burg é dada por: %0.5g',
    powerfreqbeforedsp)

[power_timebefore_yulear power_freqbefore_yulear]=
    dspmaria_yulear(x,Fs,9);
sprintf('A energia pelo método AR é dada por: %0.5g',
    powerfreqbeforedsp)
```

2.1 2a.

Using the Fourier method, the author implemented the signal's power spectrum doing the fourier spectrum and calculating the square of the magnitude. As it is possible to see in the code:

```
function [power_time power_freq]=dspmaria_fft(
    interpol_intervaloRR, Fsnew)
signal_work=interpol_intervaloRR;%(
    interpol_intervaloRRmeanINTRR)/
std(interpol_intervaloRR);
Fs_work=Fsnew;
%% FFT calculation
NN=length(interpol_intervaloRR);
p2=nextpow2(NN)+2;
NFFT=2^p2;
F = Fs_work/2*linspace(0,1,NFFT/2); %% frequency vector
X=fft(signal_work, NFFT);
figure;plot(F,2*abs(X(1:NFFT/2)));title('simple FFT')
%saveas(gcf, 'FFT','fig')
%% Power spectral density
```

```
P=abs(X(1:NFFT/2));% i do not divide by the number of
    points...matlab included poss
P=P.^2;
figure;semilogy(F,P, 'r');
hpsd=dspdata.psd(P, 'Fs', Fs_work);
Fw = hpsd.Frequencies;
plot(hpsd); title('Total Power');
%saveas(gcf, 'PSD','fig');
% Parseval theorem
power_time=(sum(signal_work.^2))/length(signal_work)
power_freq=avgpower(hpsd)/(2*length(signal_work));
powerM=2*sum(P)/(NFFT*length(signal_work)) %trapezoidal
     rule
end
```

2.2 2b.

The code of the method from Welch's periodogram:

```
function [power_time power_freq] = dspmaria_welch2(
    interpol_intervaloRR, Fsnew, windowlength)
buffer=interpol_intervaloRR; %
%buffer=(interpol_intervaloRR-meanINTRR)/std(
    interpol_intervaloRR); %xnew;
%interpolation; C
% the above mean subtraction was necessary for the real
     data, for
% detrending. here it is not necessary.
Fs_buffer=Fsnew;
NN=length(interpol_intervaloRR);
p2=nextpow2(NN); % calculate length of window
NFFT=2^p2;
window=2^(p2-1);
noverlap=window-1;
%NFFTwindow=nextpow2(window)
h=spectrum.welch('Hamming',window,100*noverlap/window);
hopts=psdopts(h,buffer);
set(hopts, 'Fs', Fs_buffer, 'SpectrumType', 'twosided',
     'centerdc', false)
```

```
hpsd = psd(h,buffer,hopts);
Fw = hpsd.Frequencies;
figure; plot(hpsd);
power_freq=avgpower(hpsd);
power_time = (sum(buffer.^2))/length(buffer)
end
```

2.3 2c.

It's important to point out that the Yule and Burg's method are two ways to estimate the AR model. The AR method is an alternative to discrete Fourier transform, and the method of choice for high-resolution spectral estimation of a short time series.

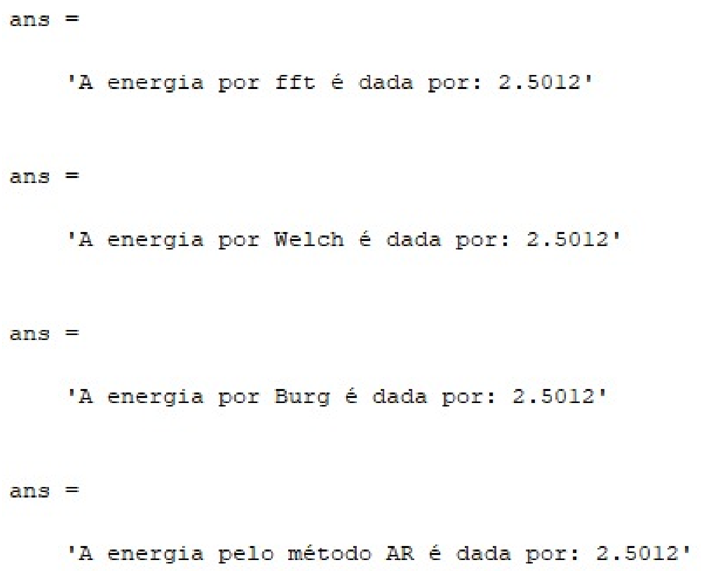
The code of the AR parametric method (Yulear):

```
function [power_time power_freq]=dspmaria_yulear(
    interpol_intervaloRR, Fsnew, p)
% [power_time power_freq]=dpsmaria_yulear(data, Fs,
    order of model)
% calculation of preservation of parseval theorem
buffer=interpol_intervaloRR; %(
    interpol_intervaloRRmeanINTRR)/
std(interpol_intervaloRR);
Fs_buffer=Fsnew; %new% %c
%% Power spectrum estimation of the interploated RR,
NN=length(buffer)
p2=nextpow2(NN);
NFFT=2^p2;
%F = Fsnew/2*linspace(0,1,NFFT/2);
h=spectrum.yulear(p);
hopts=psdopts(h,buffer);
set(hopts, 'Fs', Fs_buffer, 'NFFT', NFFT);
hpsd=psd(h,buffer, hopts);
figure; plot(hpsd);
%saveas(gcf, 'spectrumYulear', 'fig')
power_freq=avgpower(hpsd)
power_time=(sum(buffer.^2))/(length(buffer))
end
```

The code of the Burg's method:

```
function [power_time power_freq]=dspmaria_burg(
   interpol_intervaloRR, Fsnew,p)
buffer=interpol_intervaloRR; %(
   interpol_intervaloRRmeanINTRR)/
std(interpol_intervaloRR);
Fs_buffer=Fsnew; %new% %c
power_time=(sum(buffer.^2))/length(buffer)
%% Power spectrum estimation of the interploated RR, by
    Welch
NN=length(interpol_intervaloRR);
p2=nextpow2(NN); % calculate length of window
NFFT=2^p2;
F = Fsnew/2*linspace(0,1,NFFT/2);
h=spectrum.burg(p);
hopts=psdopts(h,buffer);
set(hopts, 'Fs', Fsnew, 'NFFT', NFFT)
hpsd=psd(h,buffer, hopts);
figure; plot(hpsd)
power_freq=avgpower(hpsd)
power_time=(sum(buffer.^2))/(length(buffer))
end
```

It is possible to verify in the Figure 4 that the total energy of the power spectrum calculated for each method was the same, which is 2.5012.:



```
ans =

    'A energia por fft é dada por: 2.5012'


ans =

    'A energia por Welch é dada por: 2.5012'


ans =

    'A energia por Burg é dada por: 2.5012'


ans =

    'A energia pelo método AR é dada por: 2.5012'
```

Figure 4: Power

3 Questão 3

- Link.
- Códigos adicionais e dados disponíveis na página do curso: Link.

Implemente a Parte 1 do código acima. Observe as três formas com que a densidade espectral de potência (PSD) é calculada: (1) calculando a DFT a partir do produto interno, e depois multiplicando a DFT pelo complexo conjugado, com as devidas correções de escala, para calcular o PSD; (2) calculando a DFT a partirda rotina **fft** (observe as funções **fftshift** e **periodogram**); (3) utilizando diretamente a rotina **periodogram** (digite help periodogram e veja como utilizar esta rotina corretamente). Observe que o janelamento default desta rotina é retangular.

- O resultado deve ser exatamente igual nos 3 casos. Isto aconteceu?
- Observe nos casos (1) e (2) as definições do eixo das frequências. Pelo método (3), fornecendo-se N e 1/dt, a rotina periodogram já define o eixo de frequências correto, em Hz.

```
%%  Part 1:  Power Density Spectrum
close all; clear all;

dt = 0.001;                 % The sampling interval. fs=
    1000 Hz
t = (dt:dt:3);              % Time axis. t=3s
T = max(t);                 % Max time.
N = length(t);              % Total # of data indices.
x = sin(2.0*pi*t*10) + randn(1,N);    %Sinusoid f=10Hz +
     noise.

%  Frequency axis defined between the +/- Nyquist
    frequency (fs/2)
%  in steps of df = 1/T = the frequency resolution

% Frequency axis
df = 1/T;
fNQ = 1/dt/2;
faxis = (-fNQ:df:fNQ-df);
```

3.1 Method 1: DFT calculation

The first part is made calculating the Discrete Fourier Transform iteratively:

$$X[f] = \sum x[t] \cdot e^{-2\pi f \cdot j \cdot t} \tag{4}$$

And obtaining the Power density spectrum by:

$$S[n] = \frac{2}{f_s \cdot N} X[f] \cdot X^*[f] \tag{5}$$

```
    %% Method 1: DFT calculation
%  For each freq compute the Fourier transform and save
    in X.
X = zeros(1,length(faxis));
for j=1:length(faxis)
    X(j) = sum(x .* exp(-2*pi*1i*faxis(j)*t));
```

```
end

% Power spectrum
pow = 2*dt^2/T * X.*conj(X);

%  Decibel scale.
subplot(3,1,1)
plot(faxis, 10*log10(real(pow)));  axis tight;  xlim([0
    20]);  ylim([-50 0])
grid;
xlabel('Freq [Hz]');  ylabel('Power')
```

3.2 Method 2: Built-in MATLAB routines fft and fftshift

When the DFT is performed, the signal has mirrored components on the positive and negative sides of the frequency axis. When the DFT was iteratively performed, it was not necessary to implement any further adjustments.

Although, when the fft routine is used, it is important to explicitly adjust the PSD via the fftshift routine, because the fft routine implements as default the frequencies ranging from 0 to fs and we want to visualize from $-f_s$ to f_s.

```
    %%  Method 2: Built-in MATLAB routines fft and
        fftshift
X = fft(x);
%  Power spectrum.
pow = 2*dt^2/T * X.*conj(X);

% Decibel scale and the use of "fftshift".
subplot(3,1,2)
plot(faxis, 10*log10(fftshift(pow)));  axis tight;
   xlim([0 20]);  ylim([-50 0])
grid;
xlabel('Freq [Hz]');  ylabel('Power')
```

3.3 Method 3: Built-in MATLAB routine periodogram

Obtaining the PSD via the periodogram routine is the most compact way to perform all aforementioned computations.

The inputs are the time domain vector x, the empty vector represents the rectangular window, the number of points and the sampling frequency.

```
%% Method 3: Built-in MATLAB routine periodogram
subplot(3,1,3)
periodogram(x, [], N, 1/dt);  xlim([0 20]);  ylim([-50
   0])
```

The periodogram method, implements an estimation of the power spectra dividing the time signal into successive blocks, in which w(n) represent the window used:

$$x_m(n) \triangleq w(n)x(n+mR), n = 0, 1, ..., M-1, m = 0, 1, ..., K-1 \tag{6}$$

$$P_{x_m,M}(\omega_k) = \frac{1}{M}|FFT_{N,k}(x_m)|^2 \triangleq \frac{1}{M}\left|\sum_{n=0}^{N-1} x_m(n)e^{-j2\pi nk/N}\right|^2 \tag{7}$$

$$\hat{S}_x^W(\omega_k) \triangleq \frac{1}{K}\sum_{m=0}^{K-1} P_{x_m,M}(\omega_k) \tag{8}$$

As expected, it is shown in Figure (5) that the result of the 3 methods is rigorously the same.

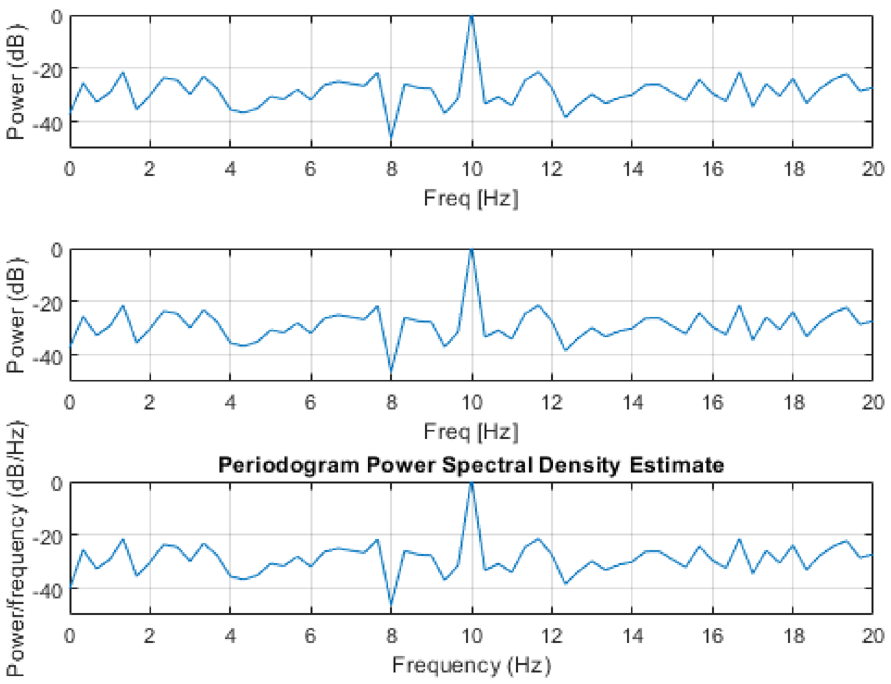


Figure 5: PSD of signal obtained by equivalent methods

4 Questão 4

(cont.) Implemente a Parte 2 ("Part 2: Compute the cross covariance (by hand) and apply to noisy data") do código anterior. Neste item, será calculada a covariância cruzada entre dois sinais aleatórios.

- O que você espera observar?

Initially, the signals x and y are made up random numbers normally distributed, because the function randn() follows a standard normal distribution. Thus, it's expected that the array of points of the signal have a mean 0 and a variance 1 with random numbers following these parameters.

In the cross-covariance, according to the definition, it is a measure of the similarity of the deviation of two signals about their respective means. In other words, the cross-covariance measure the correlation in the variation between signals. In this example, despite the signals x and y following a standard normal distribution (mean = 0 and $\sigma^2 = 1$), the numbers are random and the similarity of the deviation about their respective means is also random, which tends to generate a low cross-covariance.

The MATLAB's code of the Part 2 (unmodified):

```
%%  Part 2:  Compute the cross covariance (by hand) and
    apply to noisy data.

%   First, generate two noisy signals.
dt = 0.001;                          %The sampling interval.
t = (dt:dt:1);                       %Time axis.
x = 0.2*randn(1,length(t));          %Signal x.
y = 0.2*randn(1,length(t));          %Signal y.

%   Plot the two signals
figure(1);
subplot(2,1,1);
plot(t, x, 'b', 'DisplayName', 'Signal x', 'linewidth',
    1.3);
hold on;
plot(t, y, 'g', 'DisplayName', 'Signal y', 'linewidth',
    1);
hold off;
xlabel('Time (s)');
ylabel('Amplitude');
legend('Orientation','horizontal','Box','on','Location'
   ,'northeast');
title('Amplitude x Time - Signals x and y');

%  Computing the cross-variance
[rxy, lag] = my_cc_circ_shift(x, y);

%   And plot the results
subplot(2,1,2);
plot(lag, rxy, 'b', 'linewidth', 1.3)
xlabel('Lags (ms)');
ylabel('Cross-covariance');
title('Cross-covariance between signals x and y (r_{xy
   })');
saveas(gcf, 'q4_fig1.png');
```

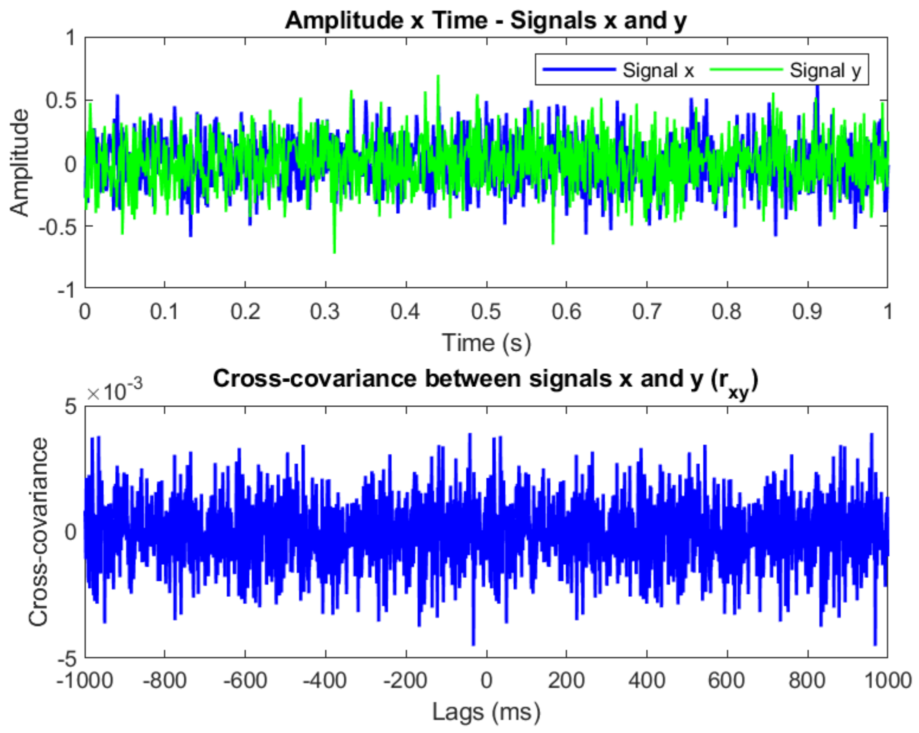


Figure 6: Random signals (normally distributed) plotted and cross-covariance between them.

- Observe que será necessário realizar o download da função "my_cc_circ_shift.m", disponível na página do curso mencionado: http://math.bu.edu/people/mak/MA666/
- Depois, compare estes resultados com o uso da rotina "xcorr" do Matlab para realizar o mesmo cálculo e verifique se o resultado deu o mesmo.

The MATLAB's code of the Part 2 (modified):

```
%% Part 2 - Modified

% Cross-covariance of x and y using the built-in
    function xcorr()
[cxy, lags_cxy] = (xcorr(x - mean(x), y - mean(y))); %
    cross-covariance
N = length(t); % num of samples
```

```
cxy = cxy ./( (N) ); % normalization - eliminate the
    bias of the num of samples

% Plot of cross-crovariance with 'my_cc_circ_shift.m'
figure(2);
subplot(2,1,1);
plot(lag, rxy, 'b', 'linewidth', 1.3)
xlabel('Lags (ms)');
ylabel('Cross-covariance');
title('Cross-covariance between signals x and y (r_{xy
    })');

% Plot of cross-covariance with 'xcorr()'
subplot(2,1,2);
plot(lags_cxy, cxy, 'b', 'linewidth', 1.3);
xlabel('Lags (ms)');
ylabel('Cross-covariance');
title('Cross-covariance (c_{xy}) with xcorr');
saveas(gcf, 'q4_fig2.png');
```

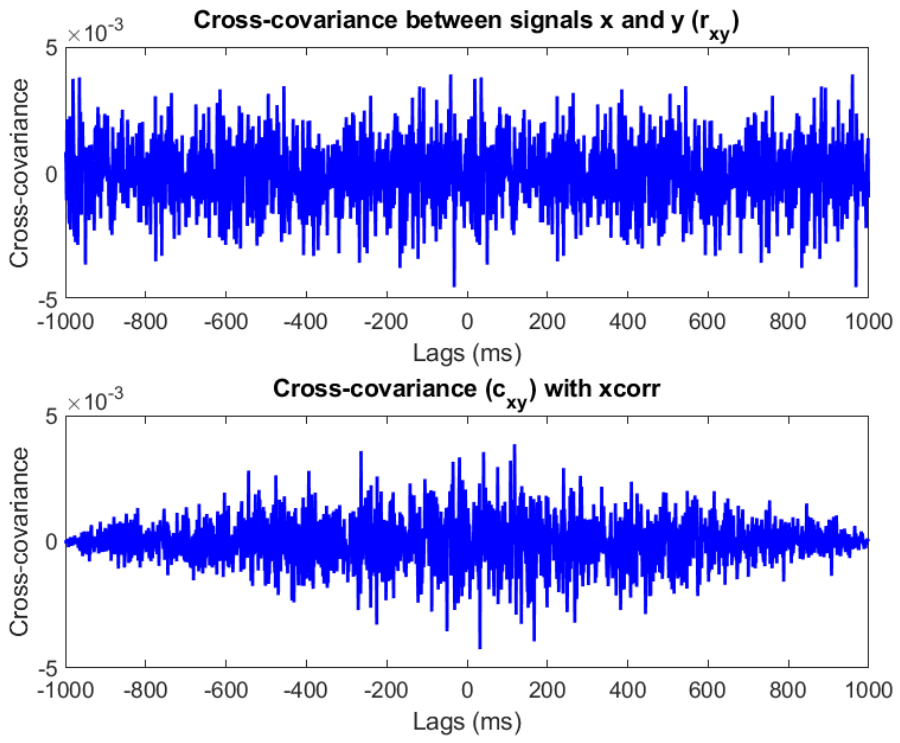


Figure 7: Comparison of cross-covariance - using the function "my_circ_cc_shift.m" and the xcorr(), respectively.

In the Figure 7, it is notorious a slightly difference between the cross-covariance applied using the "my_circ_cc_shift.m" function and the xcorr() function. The first, in its code, calculates the cross-covariance with positive, negatives and zero shifts with the function circshift(), and it shifting process generates a difference in terms of correspondent values when compared with the xcorr() function in the second case.

Despite this, the cross-covariance in both case keeps the mean around 0 and the variance around 1, also with low values, because the signal is composed by random numbers and the similarity of the deviation about their respective means is also random, which tends to generate this low value of cross-covariance.

5 Questão 5

(cont.) Implemente a Parte 3 ("Part 3: Relationship of power spectrum to autocovariance") do código. O objetivo é mostrar que a densidade espectral de potência (PSD) é a transformada de Fourier da função de auto-covariância. Da mesma forma, a função de autocovariância pode ser obtida a partir da transformada inversa de Fourier do PSD.

- Será calculada a transformada de Fourier (DFT) de um sinal senoidal com ruído branco adicionado, utilizando-se a função **fft**. A partir da DFT, será calculada a densidade espectral de potência (PSD). Finalmente, será calculada a transformada inversa de Fourier da PSD. O resultado deve dar a função de autocovariância do sinal x original. Este resultado será comparado com o valor da função de autocovariância utilizado na Parte 2 do código. O resultado deve ser o mesmo, mostrando que a transformada inversa de Fourier da PSD é a função de autocovariância do sinal (com os devidos ajustes de escala) e vice-versa.

- Observe o uso da função "fftshift" para que os atrasos (lags) negativos fiquem, de fato, no lado negativo do eixo horizontal no gráfico da autocovariância.

Applying the code, we get the following result:

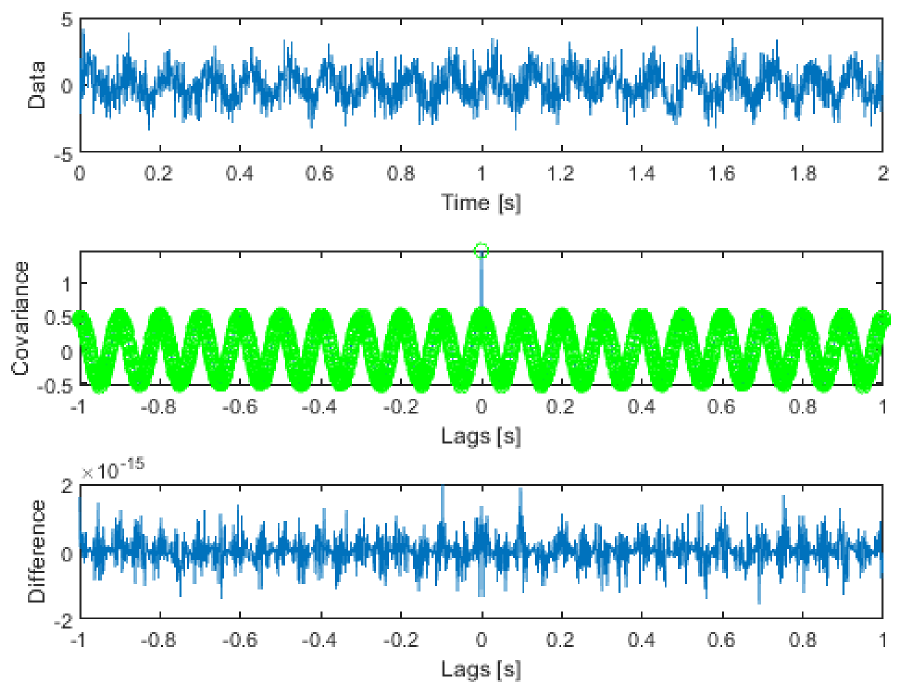


Figure 8: Autocovariance and the inverse Fourier of the PSD

As we can see, the difference measured inside matlab is very close to zero. In that way, it contributes to the fact that a signal's autocovariance is equal to the inverse Fourier of the signal's PSD and vice versa.

6 Questão 6

(cont.) Implemente a Parte 4 ("Part 4: Compute the squared coherence (by hand) between two signals") do código. O objetivo é calcular a função de coerência entre dois sinais aleatórios. $\Rightarrow$ O que você espera observar?

- Observe como é realizado o cálculo dos autoespectros de **x** (S_{xx}) e **y** (S_{yy}) e o espectro cruzado (S_{xy}), para então determinar a função de coerência.

- Qual foi o resultado da função de coerência? Isto faz sentido, uma vez que os dois sinais são aleatórios, sem nenhuma relação entre si? Por que isto ocorreu?

- Calcule agora a função de coerência entre x e y utilizando a função mscohere do Matlab: [C_{xy},**F**] = mscohere(x,y,[],[],length(x),1/dt,'centered');. Plote o resultado (utilize a opção **xlim([-50,50])** como no exemplo). O que mudou? Porque isto ocorreu? Para ajudar na sua resposta, digite "**help mscohere**" e veja as opções default da função do Matlab. Por que estas opções fazem com que a função de coerência dê diferente daquela calculada na Parte 4 do código anterior? Qual faz mais sentido com os sinais x e y utilizados neste exemplo (ou seja, qual implementação está "correta" e por quê)?

According to "Signal processing for neuroscientists, W. Van Drongelen", Coherence is a correlation analysis in the frequency domain, instead of the previous time domain methods studied. This index has the advantage of obtaining frequency-specific correlations.

As both signals are randomly generated, it is expected that they have low coherence.

```
    %% Part 4: Squared coherence between two signals.

close all; clear all;

% Define two signals. Can you guess their coherence?
```

```
dt = 0.001;  t = (dt:dt:1);  N = length(t);  T = max(t);
x = randn(1,N);                          %Signal x, random noise.
y = randn(1,N);                          %Signal y, random noise.

%  Frequency axis.
df = 1/T;
fNQ = 1/dt/2;
faxis = (-fNQ:df:fNQ-df);

% Fourier transform of x.
Xd = fft(x);

% Fourier transform of y.
Yd = fft(y);

% Auto and cross spectra, and organize frequencies.
Sxx = 2*dt/N*(Xd.*conj(Xd));  Sxx = fftshift(Sxx);
Syy = 2*dt/N*(Yd.*conj(Yd));  Syy = fftshift(Syy);
Sxy = 2*dt/N*(Xd.*conj(Yd));  Sxy = fftshift(Sxy);

% Compute the squared coherence.
cohr = Sxy.*conj(Sxy) ./ (Sxx.*Syy);

% Plot the results.
subplot(5,1,1)
plot(t,x)
hold on
plot(t,y, 'g')
hold off
xlabel('Time [s]');  ylabel('Data')

subplot(5,1,2)
plot(faxis, Sxx);  ylabel('Abs(Sxx)^2');  xlim([-50,50])
subplot(5,1,3)
plot(faxis, Syy);  ylabel('Abs(Syy)^2');  xlim([-50,50])
subplot(5,1,4)
```

```
plot(faxis, Sxy.*conj(Sxy));  ylabel('Abs(Sxy)^2');
    xlim([-50,50])
subplot(5,1,5)
plot(faxis, cohr)
xlabel('Freq [Hz]'); ylabel('Squared Coherence'); xlim(
    [-50,50]); ylim([-0.1,1.1])
```

When the provided code was executed, the expected result was not obtained, as the plot in Figure (9) suggests that the coherence is "1" between signals x and y.

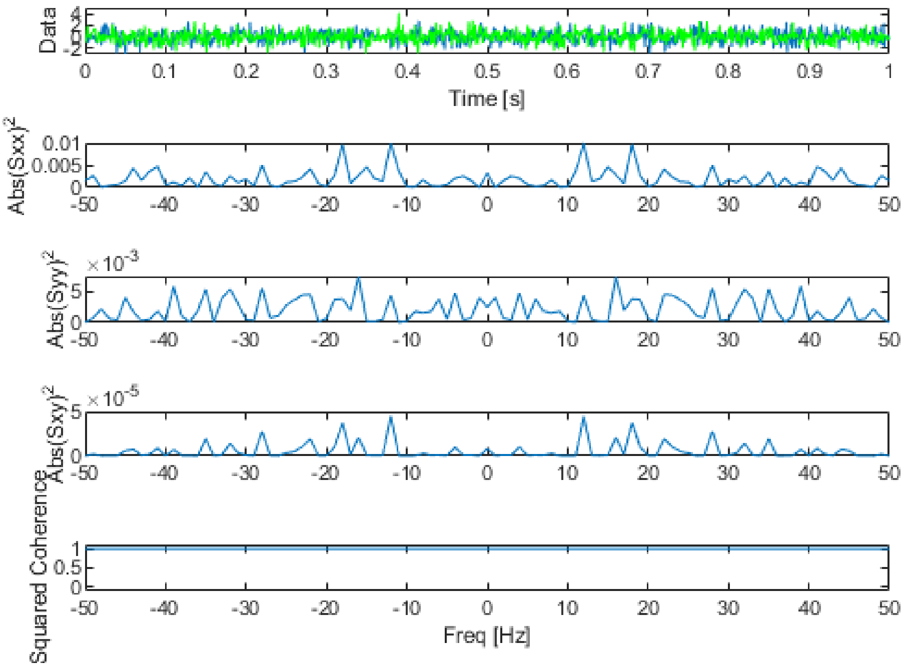


Figure 9: Provided code output for coherence between random noise signals

Further analysis shows that this result is a direct consequence of the way coherence was calculated. By the Tarefa04_TE_202002 _Matlab.pdf page 10:

$$\gamma^2 = \frac{S_{uy}(f)S^*_{uy}(f)}{S_{uu}(f)S_{yy}(f)} = \frac{Y_1(f)X^*_1(f)X_1(f)Y^*_1(f)}{X_1(f)X^*_1(f)Y_1(f)Y^*_1(f)} = 1.0 \qquad (9)$$

In order to make a useful index, it only makes sense to use coherence when some epochs are averaged.

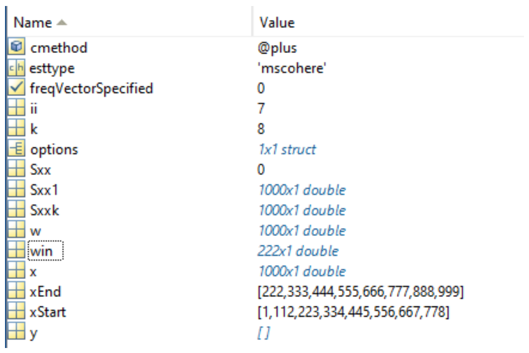
By default, the function mscohere from Matlab divide the signals x,y into 8 sections windowed with a Hamming window with 50% overlap. Every periodogram is then computed and averaged.

This can be verified analysing Matlab's internal function calls of mscohere:

```matlab
...
    Sxx1 = zeros(0,class(Sxx)); %initialization
    for ii = 1:k
         [Sxxk,w] = computeperiodogram(x(xStart(ii):xEnd(ii),:),win,...
             options.nfft,esttype,options.Fs);
         if ii == 1
             % use Sxx for applying cmethod in first iteration
             Sxx1 = cmethod(Sxx,Sxxk);
         else
             % accumulate from the second iteration
             Sxx1  = cmethod(Sxx1,Sxxk);
         end
    end
 end

Sxx1 = Sxx1./k; % Average the sum of the periodograms
...
```

The working space variables are in Figure 10.



Name	Value
cmethod	@plus
esttype	'mscohere'
freqVectorSpecified	0
ii	7
k	8
options	1x1 struct
Sxx	0
Sxx1	1000x1 double
Sxxk	1000x1 double
w	1000x1 double
win	222x1 double
x	1000x1 double
xEnd	[222,333,444,555,666,777,888,999]
xStart	[1,112,223,334,445,556,667,778]
y	[]

Figure 10: Working space of function welch.m in iteration 7/8 of Sxx1

The k number of divisions is 8 and their indexes range from 1-222, 112-

333, 223-444, 334-555, 445-666, 556-777, 667,888, 778-999. So each vector overlaps 50% elements with the previous partition. And the result is averaged by the number of partitions.

This operation is well ilustrated in Figure 11 of "Signal processing for neuroscientists, W. Van Drongelen".

The result previously obtained is altered, when the spectrum is averaged between the samples established before and metric results.

$$\gamma^2 = \frac{S_{uy}(f)S^*_{uy}(f)}{S_{uu}(f)S_{yy}(f)} = \frac{\mid < S_{uy}(f) >_n \mid^2}{< S_{uu}(f) >_n < S_{yy}(f) >_n} \tag{10}$$

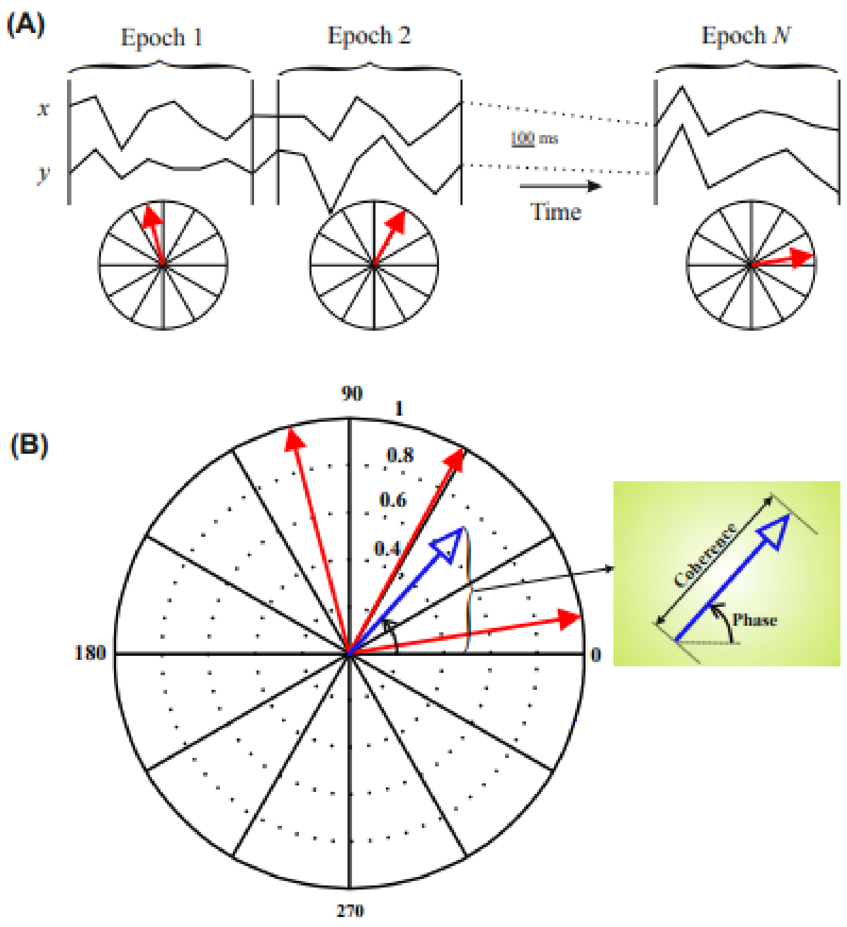


Figure 11: Extracted from "Signal processing for neuroscientists, W. Van Drongelen - Figure 13.4: Coherence computing by averaging epochs

```
    %% mscohere routine
[Cxy,F] = mscohere(x,y,[],[],length(x),1/dt,'centered')
    ;

figure;
plot(F,Cxy); xlim([-50,50]);
xlabel('Freq [Hz]'); ylabel('Coherence');
```

The output of mscohere is way more meaningful as seen on Figure (12).

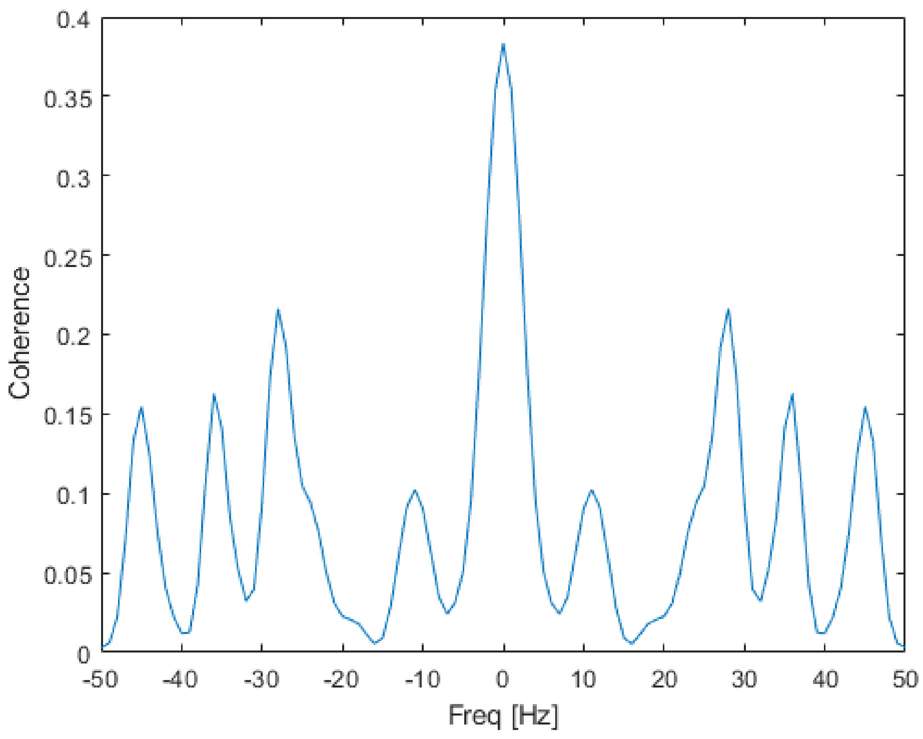
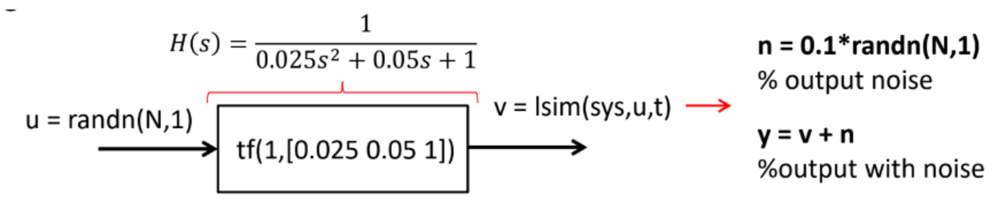


Figure 12: Coherence obtained by "mscohere" routine

As verified, the only way that coherence can provide real meaning is averaging subsequent epochs. The implementation of "mscohere" unarguably is the right implementation of this algorithm.

7 Questão 7

Em Lec3_SmoothSuu (Disponível em Link e também na página de nosso curso), o autor implementa uma função "sys", utilizando o comando "tf" (Transfer Function, ou função de transferência), e utiliza a função "lsim" para calcular a saída "v" do sistema, dada a entrada "u", como indicado na figura abaixo:



- Observe que o código primeiro implementa a função de transferência e a coerência diretamente a partir do algoritmo fft ("no smoothing"). Veja que o valor da coerência, neste caso, é sempre unitário.

- A seguir, o código calcula a função resposta em frequência (FRF) teórica, utilizando o sistema definido na variável "sys", utilizando a função "freqresp", na seção "Theoretica FRF".

- Finalmente, o código calcula a DFT pelo método de Welch. Observe que a entrada é dividida em D = 10 segmentos, e é calculada a FFT de cada segmento. A seguir, o código calcula a densidade espectral de potência de cada segmento da entrada ("SuuWall") e, na sequência, realiza a média dos D = 10 segmentos, na variável "SuuW". O mesmo é feito para a saída y (variáveis "SyyWall" e "SyyW") e para a correlação cruzada (variáveis "SuyWall" e "SuyW").

- A função de transferência estimada pelo método de Welch é calculada na variável "HW" e a coerência pelo mesmo método na variável "CW".

- Observe que, agora, os valores da coerência fazem mais sentido.

- Finalmente, o código faz os mesmos cálculos, mas utilizando uma média nas frequências (código "freqAvg.m").

- Observe que o método de Welch pode ser calculado utilizando-se a função do Matlab "pwelch".

- Para testar, implemente o mesmo exemplo utilizado em Lec3_SmoothSuu.m, com a mesma entrada "u" e a mesma saída "y", mas estimando a função de transferência $H(w)$ e a coerência Cw pelo método de Welch, utilizando a função do Matlab "pwelch".
- Veja a implementação deste método para determinar a função de transferência entre a pressão arterial e o intervalo RR (e como isto pode ser interpretado) na dissertação de mestrado do aluno Victor Hugo: Link.

The MATLAB's code:

```matlab
%% defines the system and signals

dt=0.01;            % sample frequency
fs=1/dt;            % sample frequency
N=5000;             % # of samples
T=N*dt;             % observation time
t=(0:N-1).'*dt;     % time vector
u=randn(N,1);       % input signal
sys=tf(1,[0.025 0.05 1]); %low pass system
v=lsim(sys,u,t);% output of the system
n=0.1*randn(N,1); % output noise
y=v+n;              % output with noise

%% Welch Method - Hands on (original code)

D = 10; %10 segments
u_w = reshape(u, N/D ,D);    % make matrix with D
    columns!
U_w = fft(u_w);              % fft works columnwise on
    matrices
U_w = U_w(2:N/D/2+1, :);
SuuWall = D/N*real(conj(U_w).*U_w);
SuuW = mean(SuuWall, 2);    %take the average over the
    rows (,2 option)
y_w = reshape(y, N/D, D);    % make matrix with D
    columns!
Y_w = fft(y_w);              % fft works columnwise on
    matrices
Y_w = Y_w(2:N/D/2+1, :);
SyyWall = D/N*real(conj(Y_w).*Y_w);
```

```
SyyW = mean(SyyWall, 2);
SuyWall = D/N*conj(U_w).*Y_w;
SuyW = mean(SuyWall,2);
fw = (1:N/D/2).'*D/N/dt;
HW = SuyW./SuuW; % ft by Welch Method
CW = abs(SuyW).^2./(SuuW.*SyyW); % coherence by Welch
    Method

% figure with spectra
figure(1);
subplot(311);
loglog(fw, SuuW, 'r', 'linewidth', 1.3);
xlabel('Frequency');
ylabel('S_{uu}(w)');
title('Power Spectrum - S_{uu}');

subplot(312);
loglog(fw, SyyW, 'r', 'linewidth', 1.3);
xlabel('Frequency');
ylabel('S_{yy}(w)');
title('Power Spectrum - S_{yy}');

subplot(313);
loglog(fw,abs(SuyW), 'r', 'linewidth', 1.3);
xlabel('Frequency');
ylabel('S_{uy}(w)');
title('|S_{uy}|');
saveas(gcf, 'q7_fig1.png');

% figure with FRF and coherence
figure(2);
subplot(311);
loglog(fw,abs(HW), 'r', 'linewidth', 1.3);
xlabel('Frequency');
ylabel('|H(w)|');
title('Module of Frequency Response');

subplot(312);
semilogx(fw, angle(HW)*180/pi, 'r', 'linewidth', 1.3);
xlabel('Frequency');
ylabel('<H(w)');
```

```
title('Phase of Frequency Response');

subplot(313);
semilogx(fw, CW, 'r', 'linewidth', 1.3);
xlabel('Frequency');
ylabel('C(w)');
title('Coherence');
saveas(gcf, 'q7_fig2.png');


%% applying Welch Method to input (u) and output (y)

[Suu_w, fu] = pwelch(u, N, D, [], fs); % welch of auto-
    spectrum of u
[Syy_w, fy] = pwelch(y, N, D, [], fs); % welch of auto-
    spectrum of y

[Suy_w, fuy] = cpsd(u, y , N, D, [], fs); % cross
    spectrum

HW_new = Suy_w./Suu_w; % transfer function by Welch
    Method
[CW_new, f_cw] = mscohere(u,y); % coherence by Welch
    Method

% figure with spectra
figure(3);
subplot(311);
loglog(fu, Suu_w, 'r', 'linewidth', 1.3);
xlabel('Frequency');
ylabel('S_{uu}(w)');
title('Power Spectrum - S_{uu} - Welch Method');

subplot(312);
loglog(fy, Syy_w, 'r', 'linewidth', 1.3)
xlabel('Frequency');
ylabel('S_{yy}(w)');
title('Power Spectrum - S_{yy} - Welch Method');

subplot(313);
loglog(fu, abs(Suy_w), 'r', 'linewidth', 1.3);
```

```
xlabel('Frequency');
ylabel('S_{uy}(w)');
title('|S_{uy}| - Welch Method');
saveas(gcf, 'q7_fig3.png');

% figure with FRF and coherence
figure(4);
subplot(311);
loglog(fu, abs(HW_new), 'r', 'linewidth', 1.3)
xlabel('Frequency');
ylabel('|H(w)|');
title('Module of the Frequency Response - Welch Method'
    );

subplot(312);
semilogx(fu, angle(HW_new)*180/pi, 'r', 'linewidth',
    1.3);
xlabel('Frequency');
ylabel('<H(w)');
title('Phase of the Frequency Responde - Welch Method')
    ;

subplot(313);
semilogx(f_cw, CW_new, 'r', 'linewidth', 1.3);
xlabel('Frequency');
ylabel('C(w)');
title('Coherence - Welch Method');
saveas(gcf, 'q7_fig4.png');
```

The Figures plotted before applying the Welch Method (pwelch() function) were:

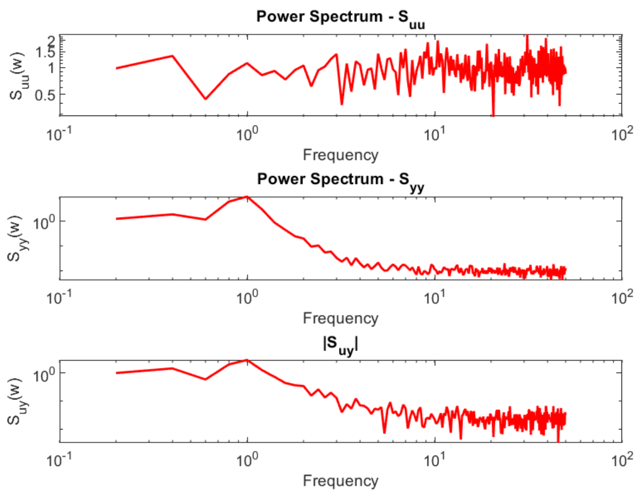


Figure 13: Each power spectrum of the signals u (S_{uu}) and y (S_{yy}), and module of the cross power spectrum between u and y (S_{uy}) - calculated before Welch Method.

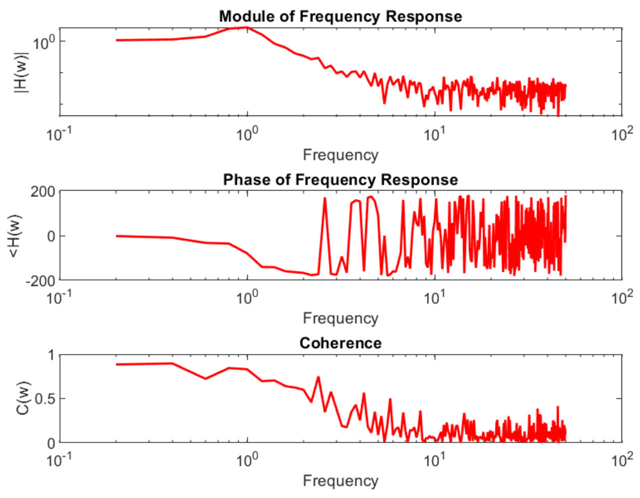


Figure 14: Module and Phase of the frequency response, and coherence - calculated before Welch Method.

Applying the Welch Method (pwelch() function), the Figures plotted were:

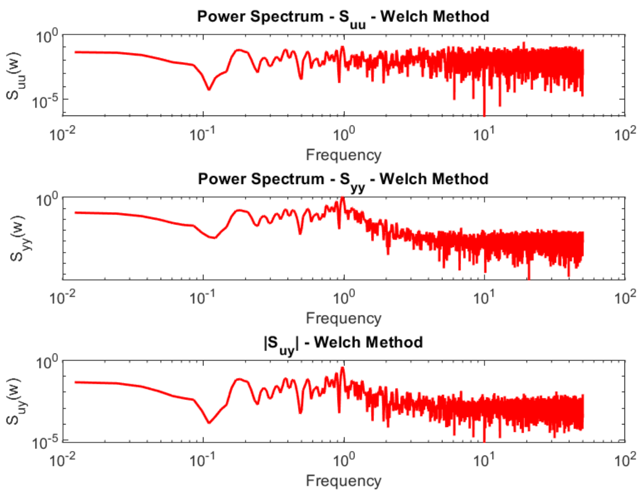


Figure 15: Module and Phase of the frequency response, and coherence - calculated after Welch Method.

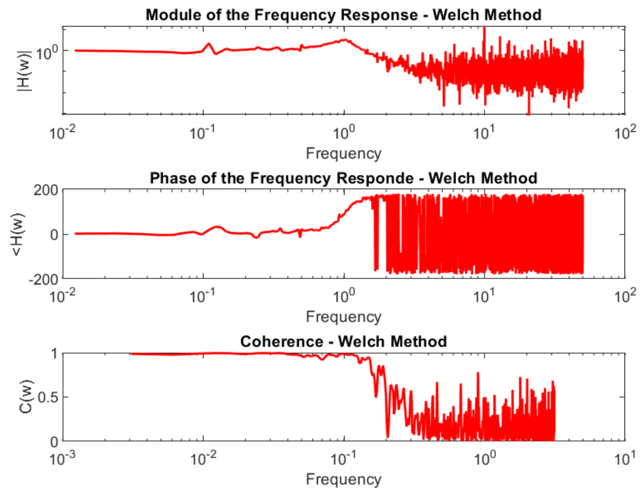


Figure 16: Module and Phase of the frequency response, and coherence - calculated after Welch Method.

The parameters of pwelch() function implemented were: $[S_{signal}$, f] = pwelch(signal, N, D, [], fs).

Which N is the numbers of samples defined as the window, and D is the number of segments defined before the Welch Method applying. The parameters were kept in order to compare the effect of the Welch Method hands-on, already applied, and the Welch Method by the pwelch() function.

According to the Figures above, evidently, the Transfer Function and the Coherence in the both cases are extremely similar. The slightly difference is probably a consequence of the use of the fft(), and its specificities of construction, in the hands-on (first case) case and the use of fs (100 Hz) in the pwelch() case. Visually, the pwelch functions provides a better and more stable visualization of the values of coherence and of the transfer function around the frequencies of interest.