Felipe Lima Alcântara

Processamento de sinais Biológicos

Prova 1 de Processamento de Sinai Biológicos, 02/2019.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Brasília, DF

09 de novembro de 2019

Sumário

1 Questão 1

1.1 Item A

Representar sinais adquiridos a uma taxa constante têm uma ineficiência inerente. A taxa deve ser suficiente para representar os eventos de maior banda, ainda que sejam esporádicos esta taxa é em geral mais alta do que seria nescessário para maior parte do sinal.

A função de uma representação multitaxa, a partir de um sinal adquirido a uma taxa constante é passarmos a representar as bandas de alta frequência a uma taxa maior e as bandas de baixa frequência a uma taxa menor, com uma gradação entre esses extremos. Por tanto se torna necessário a decomposição do sinal em diferentes bandas de frequência tornando-se necessário a filtragem inicial.

1.2 Item B

Para que haja reconstrução perfeita do sinal é necessário que a equação 1.1 seja cumprida com $A \neq 0$ e d ϵ $\mathbf{Z}$.

$$y[n] = A * x[n - d] \tag{1.1}$$

Por conta dos filtros causais haverá atraso e poderá haver uma diferença na amplitude entre a entrada e saída. Para determinar as condições a melhor forma é calcular $Y(z)$, Figura 1,sabendo que o efeito dos sub-amostradores e sobre-amostrados, de quantidade M, no domínio Z são dados respectivamente, pelas equações 1.2 e 1.3.

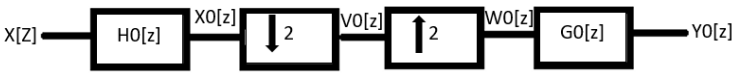


Figura 1 – Diagrama de decomposição e síntese passa baixa.

$$Y[z] = \frac{1}{M} \sum_{k=0}^{M-1} x(z^{\frac{1}{M}} W_M^{-k}) \tag{1.2}$$

$$Y[z] = X[z^M] \tag{1.3}$$

Iniciamos calculando a saída do primeiro bloco, equação 1.4. Após calculamos a do bloco de *downsample*, equação 1.5

$$X_0[z] = X[z]H_0[z] \tag{1.4}$$

$$V_0[z] = \frac{1}{2}(X_0[z^{1/2}] + X_0[-z^{1/2}]) \tag{1.5}$$

Substituindo eq. 1.4 em eq. 1.5, encontra-se a equação 1.6.

$$V_0[z] = \frac{1}{2}(X[z^{1/2}]H_0[z^{1/2}] + X[-z^{1/2}]H_0[-z^{1/2}]) \tag{1.6}$$

Após isso é calculada a saída do bloco de *upsample* e substituída a eq. 1.6 no resultado, equação 1.7.

$$W_0 = V[z^2] = \frac{1}{2}(X[z]H_0[z] + X[-z]H_0[-z]) \tag{1.7}$$

Por fim encontra-se a saída, equação 1.8.

$$Y_0[z] = W_0[z]G_0[z] = \frac{1}{2}(X[z]H_0[z]G_0[z] + X[-z]H_0[-z]G_0[z]) \tag{1.8}$$

Analogamente,

$$Y_1[z] = W_1[z]G_1[z] = \frac{1}{2}(X[z]H_1[z]G_1[z] + X[-z]H_1[-z]G_1[z]) \tag{1.9}$$

Por fim é realizada a soma das duas saídas para formar o sinal sintetizado, equação 1.11.

$$Y[z] = Y_0[z] + Y_1[z] \tag{1.10}$$

$$Y[z] = \frac{1}{2}\{X[z]\cdot(H_0[z]G_0[z]+H_1[z]G_1[z])+X[-z]\cdot(H_0[-z]G_0[z]+H_1[-z]G_1[z])\} \tag{1.11}$$

A parcela que multiplica $X[-z]$ deve ser zerada pois representa a sobreposição do sinal, por tanto pode-se chegar nas 2 condições para reconstrução perfeita descritas nas equações 1.12 e 1.13.

$$H_0[z]G_0[z] + H_1[z]G_1[z] = 2Az^{-d} \tag{1.12}$$

$$H_0[-z]G_0[z] + H_1[-z]G_1[z] = 0 \tag{1.13}$$

1.3 Item C

Utilizando as equações 1.12 e 1.13, porém invertendo os filtros de síntese com o de decomposição, encontra-se as equações 1.14 e 1.15.

A primeira é inalterada, pois a ordem dos fatores não altera o resultado, já a segunda os índices que tiveram sinal alternado anteriormente na equação 1.13, foram alternados em G.

$$G_0[z]H_0[z] + G_1[z]H_1[z] = 2Az^{-d} \tag{1.14}$$

$$G_0[-z]H_0[z] + G_1[-z]H_1[z] = 0 \tag{1.15}$$

Logo se um banco QMF é de reconstrução perfeita não importa a ordem dos filtros sendo necessário respeitar os pares.

1.4 Item D

Para identificar o tipo de filtro foi utilizado o script abaixo.

```
%Classifing filters
    close all; clc;
    Ha = [-0.3327 0.8069 -0.4599 -0.1350 0.0854 0.0352];
    Hb = [0.0352 -0.0854 -0.1350 0.4599 0.8069 0.3327];
    Ga = [0.3327 0.8069 0.4599 -0.1350 -0.0854 0.0352];
    Gb = [0.0352 0.0854 -0.1350 -0.4599 0.8069 -0.3327];

    z_Ha = roots(Ha);
    z_Hb = roots(Hb);
    z_Ga = roots(Ga);
    z_Gb = roots(Gb);

    figure;
    subplot(221);zplane(z_Ha);
    title('Ha');
    subplot(222);zplane(z_Hb);
    title('Hb');
    subplot(223);zplane(z_Ga);
    title('Ga');
    subplot(224);zplane(z_Gb);
    title('Gb');
```

Primeiramente encontrou-se o valor dos zeros e assim pode-se plotar os pontos no domínio Z, Figura 2. Desta forma já é possível determinar visualmente o tipo de filtro, porém ainda foi utilizada a equação 1.16, para determinar os ganhos e assim provar o que foi notado visualmente, a respostas se encontra na Tabela 1.

$$|H(jw)| = \prod_{k=0}^{M} |e^{jw} - z_k| \tag{1.16}$$

Para determinar a reconstrução perfeita foi utilizado o função feita em sala denominada `evaluate_decomposition_synthesis_filters`, inicialmente é realizada a criação

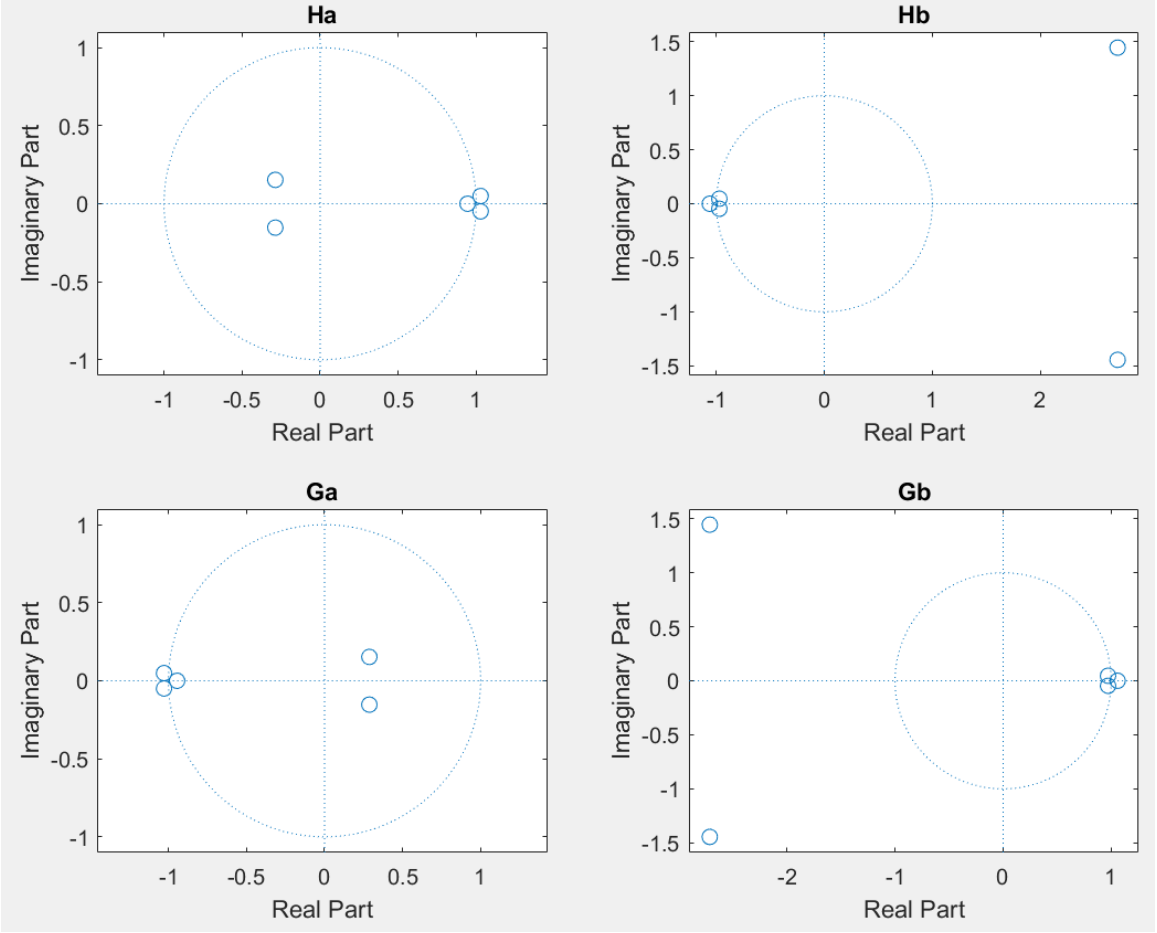


Figura 2 – Zeros dos filtros da questão 1 D no Domínio Z.

Tabela 1 – Respostas em frequência dos Filtros.

Filtros	Classificação	Local QMF
H_a	Passa-alta	H_1
H_b	Passa-baixa	H_0
G_a	Passa-baixa	G_0
G_b	Passa-alta	G_1

dos filtros com coeficientes negativos($H[-z]$), estes terão seus coeficientes múltiplos de 2 alternados, isto é realizado na função `alternate_coefficients_signals`.

Após a alternância do sinal dos coeficientes, basta-se realizar a soma dos poliônimos da convolução dos filtros passa-baixas entre si e passa-altas analogamente, como os filtros podem ter tamanhos diferentes é necessário antes igualar seus tamanhos para isso foi criada a função `polynomial_sum`.

Por fim basta analisar se os resultados cumprem as equações 1.12 e 1.13, nota-se que foi estabelecido uma tolerância miníma para o valor de A, A deve ser maior que `tol/2`.

1.4.1 evaluate_decomposition_synthesis_filters

```
function [perfect_reconstruction, A, d] = ...
    evaluate_decomposition_synthesis_filters(h0, h1, g0, g1, tol);
    if ¬exist('tol')
        tol = 1e-4;
    end
    h0_ = alternate_coefficients_signals(h0);
    h1_ = alternate_coefficients_signals(h1);
    alias_term = polynomial_sum(conv(h0_, g0), conv(h1_, g1));
    perfect_reconstruction = ¬(any(¬(abs(alias_term) < tol)));
    lti_term = polynomial_sum(conv(h0, g0), conv(h1, g1));
    k = find(abs(lti_term) ≥ tol);
    perfect_reconstruction = ((length(k) == 1) & ...
        perfect_reconstruction);
    if ¬perfect_reconstruction
        k = find(abs(lti_term) == max(abs(lti_term)));
        k = k(1);
    end
    A = lti_term(k) / 2;
    d = k - 1;
end

function h_ = alternate_coefficients_signals(h);
    h_ = h;
    h_(2 : 2 : length(h_)) = -h_(2 : 2 : length(h_));
end

function z = polynomial_sum(x, y);
    x1 = zeros(max([length(x); length(y)]), 1);
    y1 = x1;
    x1(1 : length(x)) = x;
    y1(1 : length(y)) = y;
    z = x1 + y1;
    if(size(x, 2) > size(x, 1))
        z = z.';
    end
end
```

1.5 Item E

Para determinar o tipo de decomposição do filtro basta realizar o produto interno entre os filtros de decomposição, se $H_0 \cdot H_1 = 0$ ela será ortogonal, realizando-a foi possível determinar a ortogonalidade $H_a \cdot H_b = -5.2042 * 10^{-18} \approx 0$.

1.6 Item F e G

As letras foram feitas em uma única imagem, para facilitar visualização. A Figura 3 demonstra um banco QMF sem considerar o atraso, já a Figura 4 considera o atraso dos blocos, o d utilizado foi encontrado pela função `evaluate_decomposition_synthesis_filters`.

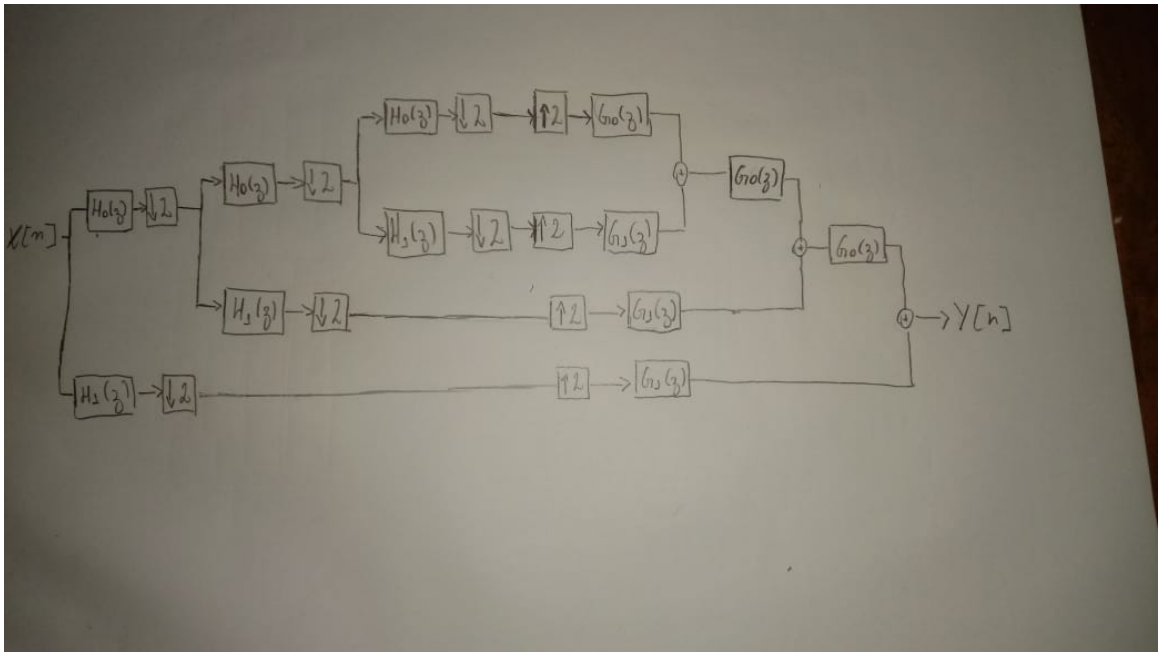


Figura 3 – Banco QMF com 3 níveis.

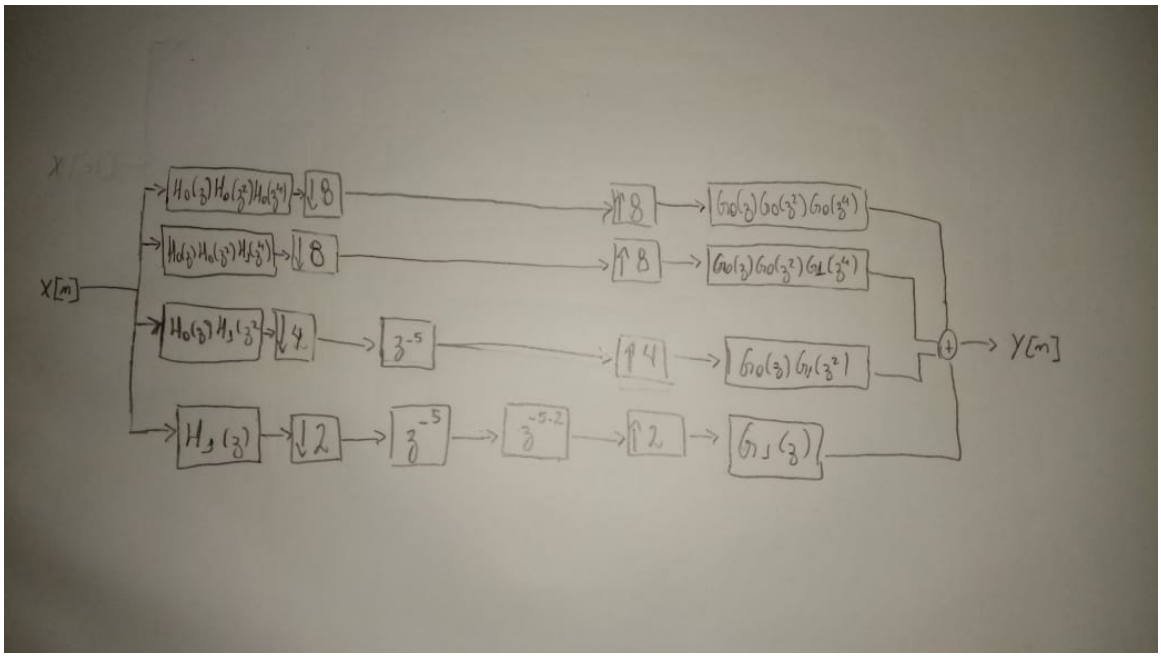
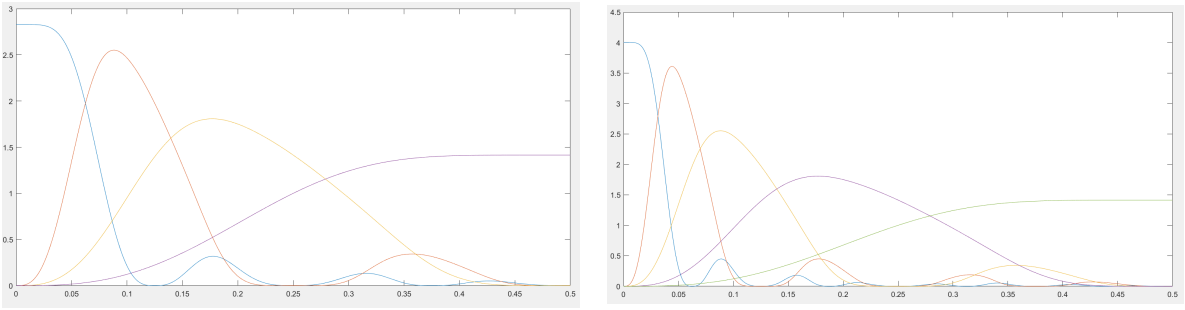


Figura 4 – Banco QMF com 3 níveis considerando um atraso em cada estágio.

1.7 Item H



(a) 3 níveis. (b) 4 níveis.

Figura 5 – Resposta em frequência dos filtros de decomposição.

1.8 Item I

Uma decomposição multinível QMF utiliza pequenas janelas em baixas frequência e grandes em baixa frequência, por tanto a resolução no tempo das altas frequências enquanto a resolução em frequência se torna melhor em baixas frequências.

Ao se realizar uma representação *tempo x frequência* resulta-se na imagem a direita da Figura 6. Nota-se que em altas frequências o resultado é maior verticalmente, isso é baixa resolução em frequência, porém horizontalmente é menor, resultando em uma melhor resolução no tempo.

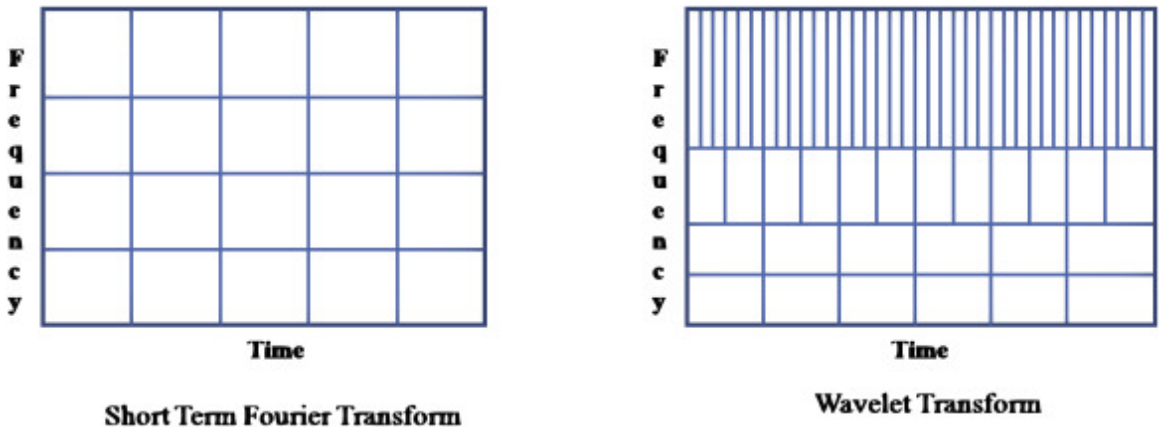


Figura 6 – Representação Tempo-Frequência com STFT ou WT.

1.9 Item J

A existência de diversos filtros está diretamente ligada a necessidade de aplicação em diversos tipos de sinais, estes que precisam de um processo de processamento diferente,

para se obter as informações. Ortogonalidade, linearidade de fase e forma da resposta em frequência.

1.10 Item K

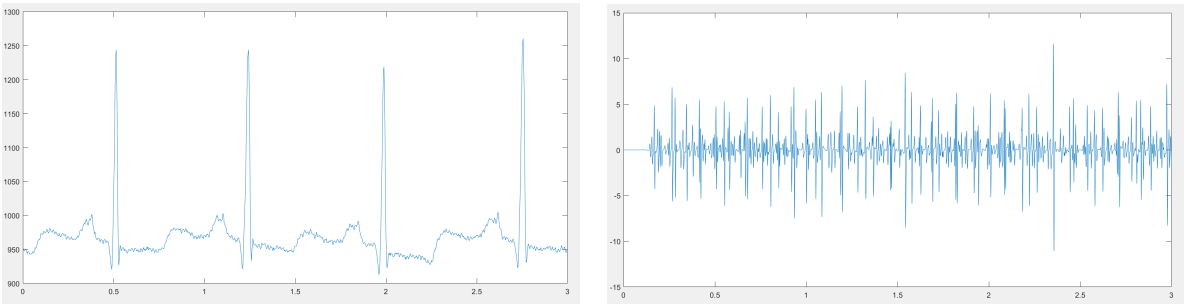
Filtros ortogonais na decomposição, formam um conjunto de bases que descrevem o sinal, e podem ser utilizados na decomposição e reconstrução, os filtros de fase linear tem um atraso em frequência constante, assim não existe uma necessidade de se adequar o filtro a cada tipo de frequência.

Tabela 2 – Características das transformadas *wavelets*.

	Daubechies	Coiflets	Symlets	Biortogonais
Ortogonal	Sim	Sim	Sim	Não
Linear	Não	Sim	Não	Não

2 Questão 2

Os sinais utilizados na questão foram os apresentados na Figura 7.

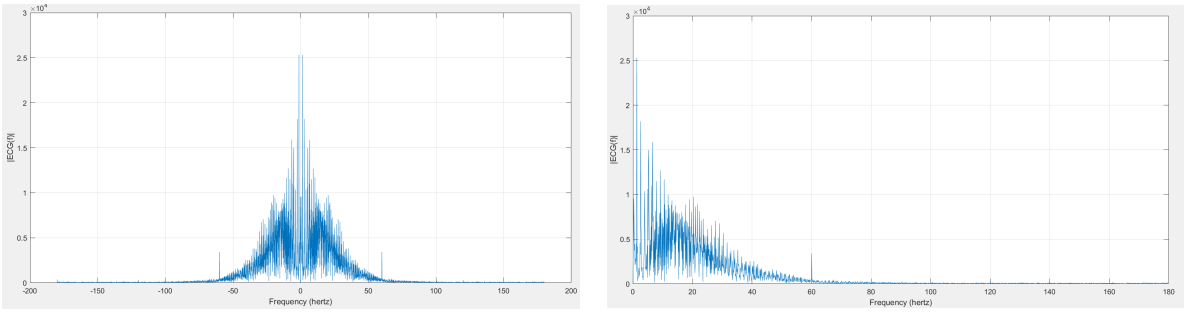


(a) Eletrocardiograma (b) Eletromiograma.

Figura 7 – Gráficos Tempo(s) x Tensão(mV).

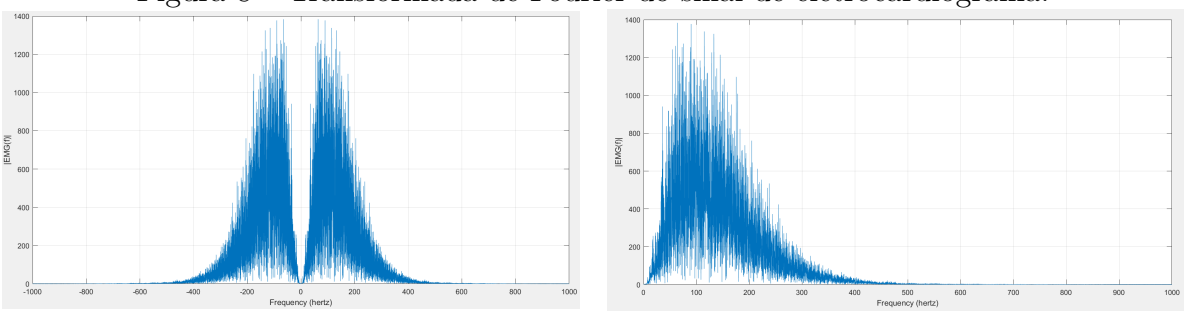
2.1 Item A

As Figuras 8 e 9, são as transformadas de Fourier do sinal de eletrocardiografia e de eletromiografia, respectivamente. Para a apresentação das imagens foi criado o script `quest2a.m`.



(a) Frequências positivas e negativas.. (b) Apenas frequências positivas.

Figura 8 – Transformada de Fourier do sinal de eletrocardiograma.



(a) Frequências positivas e negativas. (b) Apenas frequências positivas.

Figura 9 – Transformada de Fourier do sinal de eletromiograma.

```
%quest2a.m
    close all; clc;
    ecg_1 = load('ecg_1.mat', 'x','fs');
    emg_1 = load('emg_1.mat', 'x','fs');

    ecg = ecg_1.x;
    fs_ecg = ecg_1.fs;
    emg = emg_1.x;
    fs_emg = emg_1.fs;
    ecg = ecg-mean(ecg);
    emg = emg-mean(emg);

    ECG = fft(ecg);
    ECG = ECG(1 : round(length(ECG)/2));
    f_ecg =linspace(0, 0.5, length(ECG))*fs_ecg;
    figure;
    plot(f_ecg, abs(ECG));
    xlabel('Frequency (hertz)')
    ylabel('|ECG(f)|')
    grid on
    a = axis();

    EMG = fft(emg);
    EMG = EMG(1 : round(length(EMG)/2));
    f_emg = linspace(0, 0.5, length(EMG))*fs_emg;
    figure;
    plot(f_emg, abs(EMG));
    xlabel('Frequency (hertz)')
    ylabel('|EMG(f)|')
    grid on
    a = axis();
```

2.2 Item B

Realizando a comparação entre as Figuras 8b e 9b, nota-se que o eletrocardiograma tem uma concentração maior em uma faixa de frequências baixas, entre 2Hz à 40Hz, isto é notado pela forma do sinal no domínio do tempo, Figura 7a, onde nota-se oscilações lentas.

Já o eletromiograma, tem uma concentração em uma faixa de frequência mais alta, entre 35Hz à 240Hz, isso é notado pelas oscilações rápidas no tempo, Figura 7b

2.3 Item C e D

Para determinar o potencial de energia no sinal utilizou-se a função `bandpower` disponível no Matlab, esta função tem como saída a frequência media do sinal numa determinada banda, para isso a função usa um periodograma modificado para determinar a potência média no intervalo de frequência. O *script* utilizado para a determinação do percentual segue abaixo e os resultados se encontrão na Tabela 3.

```
%band energy percent script
ecg_1 = load('ecg_1.mat', 'x','fs');
emg_1 = load('emg_1.mat', 'x','fs');
ecg = ecg_1.x;
fs_ecg = ecg_1.fs;
emg = emg_1.x;
fs_emg = emg_1.fs;
ecg = ecg-mean(ecg);
emg = emg-mean(emg);

% Calculating the frequency band of signals
ECG = fft(ecg);
ECG = ECG(1 : round(length(ECG)/2));
f_ecg =linspace(0, 0.5, length(ECG))*fs_ecg;
EMG = fft(emg);
EMG = EMG(1 : round(length(EMG)/2));
f_emg = linspace(0, 0.5, length(EMG))*fs_emg;

%Calculating the band energy percentage
power_band_ecg = bandpower(ecg,fs_ecg,[2 40]);
total_power_ecg = bandpower(ecg,fs_ecg,[0 max(f_ecg)]);
percentage_power_ecg = 100*(power_band_ecg/total_power_ecg);
power_band_emg = bandpower(emg,fs_emg,[2 150]);
total_power_emg = bandpower(emg,fs_emg,[0 max(f_emg)]);
percentage_power_emg = 100*(power_band_emg/total_power_emg);
```

Tabela 3 – Porcentagem de energia nas faixa de frequência

Sinal(Banda)	Porcentagem de Energia
ECG$(2-40Hz)$	88.01%
EMG$(2-150Hz)$	74.45%

Nota-se com os resultados presentes na Tabela 3, que enquanto grande parte da energia do sinal de ECG está presente em uma banda pequena de frequências, 2 à 40Hz, o EMG, com mais do triplo da faixa do ECG, tem uma porcentagem menor de energia, pois ele tem sua energia dividida com mais igualdade entre as frequências.

2.4 Item E

A interferência mais fácil de se visualizar é a causada pela rede de alimentação, em 60Hz como mostrado na Figura 10. Mais difícil de se visualizar no domínio de Fourier, por estar próxima as frequências características do sinal tem-se o desvio da linha de base causada por sinais de baixa frequência gerados pelo paciente e pelos eletrodos,também estão descritos demonstrados na Figura 10. E por fim tem-se o nível DC adicionado ao sinal por conta das bioimpedâncias e das impedâncias provenientes do método de aquisição, este é retirado com uma simples subtração da media do sinal, Figura 7a, fixando assim o eixo do sinal em 0 como pode ser visto na Figura 11b.

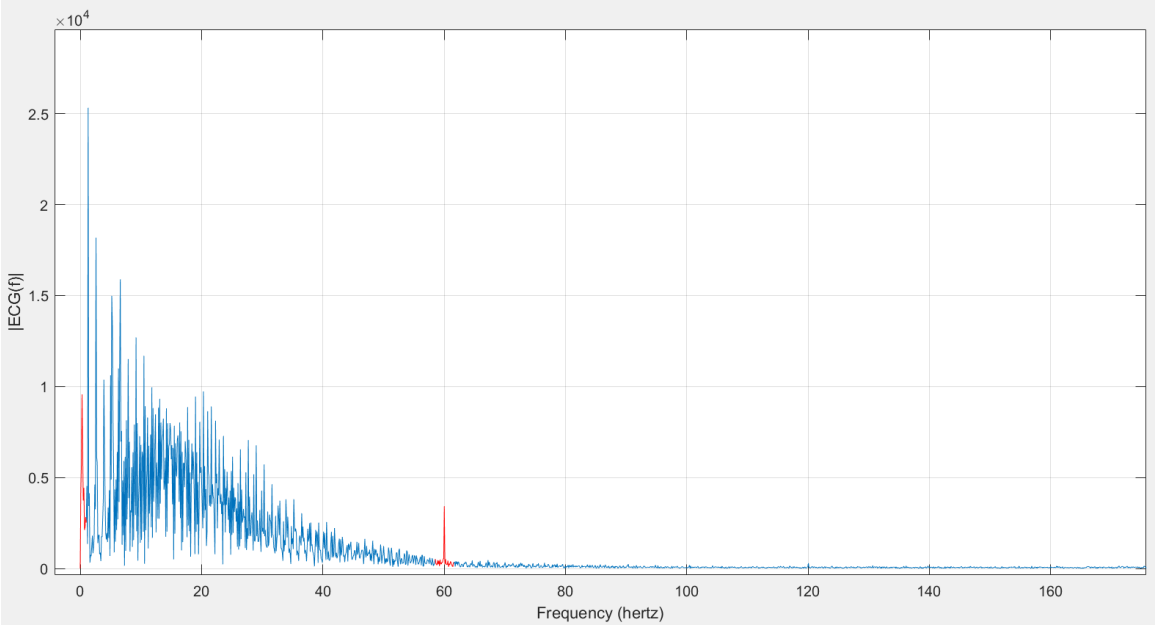


Figura 10 – Transformada de Fourier do ECG com ruídos sublinhados em vermelho.

2.5 Item F

Para tratar o sinal no domínio de Fourier utilizou-se uma função que realiza os cálculos relacionados aos índices de frequência, a filtragem do sinal e a transformada inversa, denominada `frequency_domain_filter`, secção 2.5.1, e os resultados estão explicitados na secção 2.5.2 .

2.5.1 frequency_domain_filter.m

```
function ...
    [time_filtered_signal,frequency_filtered_signal,pos_k,neg_k] = ...
    frequency_domain_filter(x,fs,nfilters,freqs,nfft)
    x= x-mean(x);
    if ~exist('nfft')
        nfft = length(x);
    end
    if length(freqs)< (nfilters*2)
        str0 = string(nfilters);
        str1 = string(length(freqs));
        error('Error. \nNumber of frequencies must be nfilters*2.\n ...
            nfilters = %s and number of freqs = %s',str0,str1)
    end
    X = fft(x,nfft);
    N = length(X);
    aux = 0;
    pos_k = zeros(1,nfilters*2);
    neg_k = zeros(1,nfilters*2);
    for n = 1:nfilters
        fc1 = freqs(n+aux);
        fc2 = freqs(n*2);
        pos_k(n+aux) = round((fc1*N/fs) + 1);
        pos_k(n*2) = round((fc2*N/fs) + 1);
        neg_k(n+aux) = round(((fs-fc2)*N/fs) + 1);
        neg_k(n*2) = round(((fs-fc1)*N/fs) + 1);
        X(pos_k(n+aux):pos_k(n*2)) = 0;
        X(neg_k(n+aux):neg_k(n*2)) = 0;
        aux = aux + 1;
    end
    frequency_filtered_signal = X;
    time_filtered_signal = ifft(X,'symmetric');
    time_filtered_signal = time_filtered_signal(1:length(x));
end
```

Inicialmente é retirada o ganho médio do sinal, assim eliminando o DC, então realiza-se a `fft` deste para o inicio da filtragem.

É necessário que se realize o calculo dos índices de frequência que estão relacionados as frequências desejadas, para isso é transformada a frequência em índice utilizando a equação 2.1, sendo f_i as frequências desejada, N o numero de amostras e f_s a frequência de amostragem. Como em Matlab os vetores se iniciam em 1, é necessário somar 1 ao resultado, os índices k encontrados estão localizados na Tabela 4.

$$k_i = \frac{f_i * N}{f_s} \tag{2.1}$$

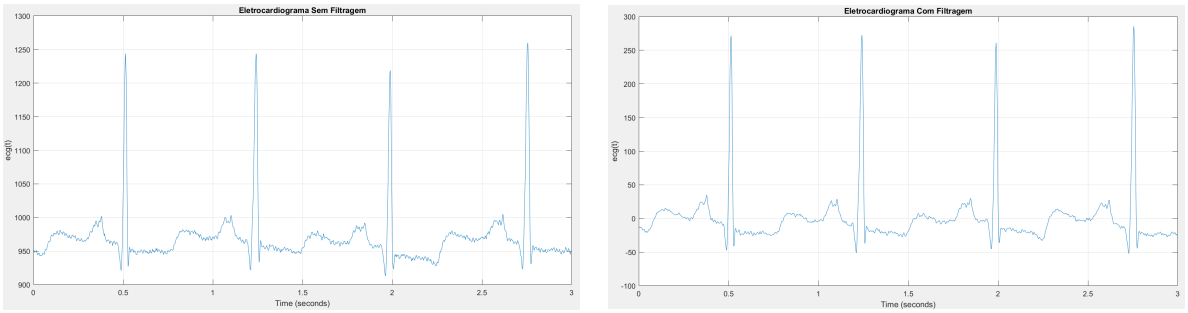
Por fim são zerados os valores dos índices.

Tabela 4 – Indices k encontrados para as frequências a serem filtradas.

Ind.\ Freq.	0Hz	1Hz	58.5Hz	61.5Hz	101Hz	180Hz
k_{pos}	1	11	586	616	1011	1801
k_{neg}	3591	3601	2986	3016	1801	2591

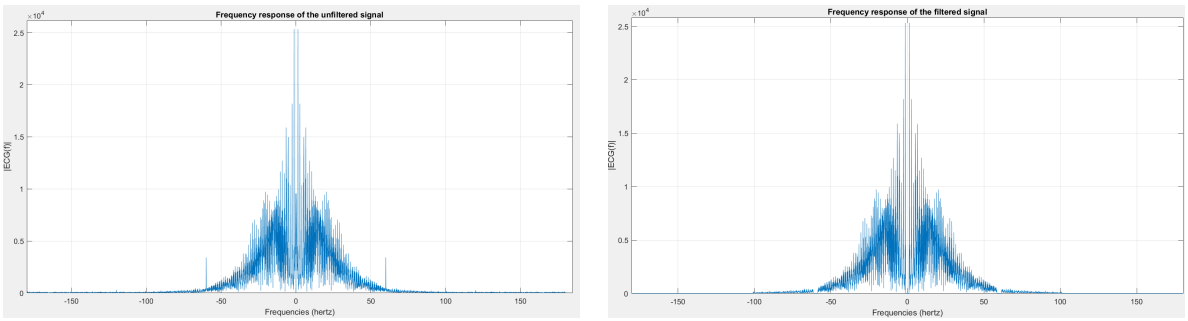
2.5.2 Resultados de Filtragem

Na Tabela 4 estão os valores de frequências filtrados, os 2 últimos valores da banda de filtragem foram colocados, pois além de ser frequências que já não tem informações utilizadas pelo ECG, dentro da faixa tem 2 harmônicas de 60Hz. Com a Figura 11 é possível comparar o resultado da filtragem no domínio do tempo, já a Figura 12 tem a comparação no domínio de Fourier.



(a) Sinal sem filtragem. (b) Sinal com filtragem.

Figura 11 – Gráficos Tempo(s) x Tensão(mV).



(a) Sinal sem filtragem(Sem DC). (b) Sinal com filtragem.

Figura 12 – Transformadas de Fourier do eletrocardiograma sem e com filtragem.

3 Questão 3

3.1 Item A

O sinal no domínio do tempo está plotado na Figura 13 e sua transformada de Fourier, retirando o ganho médio, está na Figura 14. Estas plotagens foram realizadas pelo *script* `quest3a.m`.

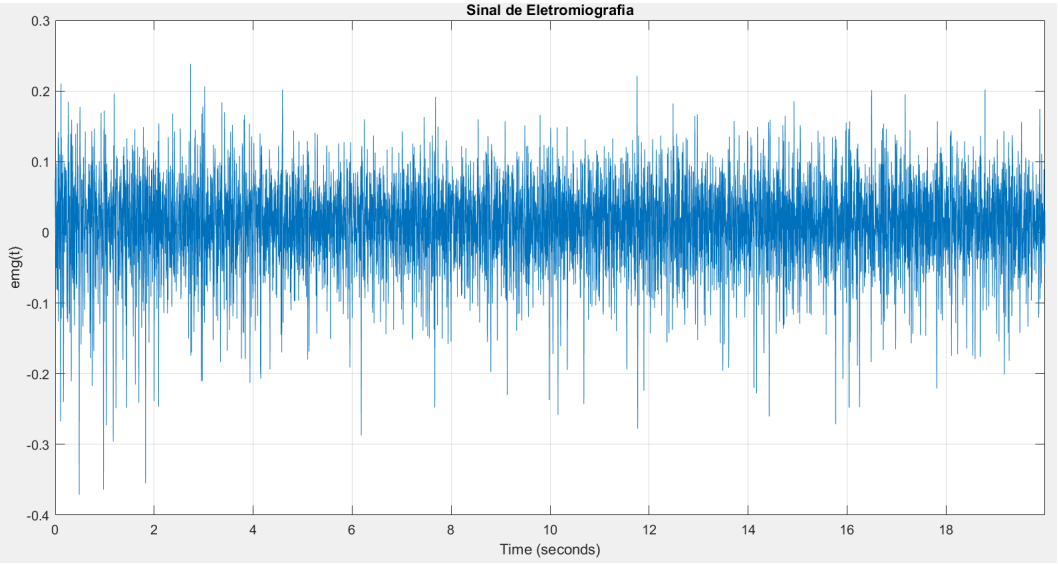


Figura 13 – Gráficos Tempo(s) x Tensão(mV) do Eletromiograma, `emg1.mat`.

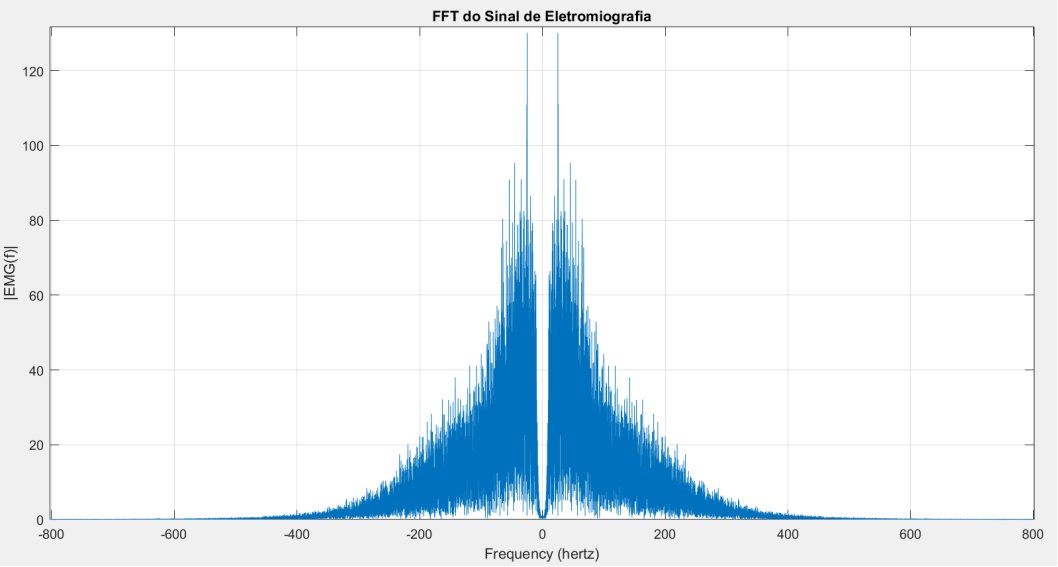


Figura 14 – Transformada de Fourier do Eletromiograma.

```
%quest3a.m
close all; clc; clear all;
load('emg1.mat', 'x','fs');
t = 0 : (1/fs) : length(x)/fs;
t = t(1 : length(x));
x = x-mean(x);
window_duration = 0.1;
figure;
plot(t, x);
xlim([0 max(t)])
xlabel('Time (seconds)');
ylabel('emg(t)');
title('Sinal de Eletromiografia')
grid on;

X = fft(x);
f =linspace(-0.5, 0.5, length(X))*fs;
figure;
plot(f, fftshift(abs(X)));
xlabel('Frequency (hertz)')
ylabel('|EMG(f)|')
title('FFT do Sinal de Eletromiografia')
grid on
```

3.2 Item B

Para esta questão foi criada a função `noising_signal`, ela cria n sinais de acordo com o numero de eventos a serem inseridos, duração, frequências e energia. Retornando o sinal com ruído aplicado e os índices onde os ruídos foram aplicados.

3.2.1 `noising_signal.m`

Inicialmente janela-se o sinal para identificação da energia média, esta que é realizada em seguida com uma simples função `min_win_energy`.

As frequências possíveis para as senoides são calculadas e são alocadas em um vetor para serem enviada em seguida para a função `make_sinusoidal_signal`, que gerará sinais compostos pela soma de diversas senoides com amplitudes randômicas.

Por fim o ruido é aplicado no sinal pela função `noise_application`, que insere os eventos em locais aleatórios do sinal espaçados de por 2 vezes o numero de eventos.

```
function [noise_emg,noise_input]= ...
    noising_signal(x,fs,window_duration,...
     events, min_freq,max_freq,min_energy);
     [windowed_signal] = windowing(x,window_duration,fs);
     [energy] = min_win_energy(windowed_signal,min_energy);
     freqs_in_range = min_freq : 0.05: max_freq;
     aux = 0;
     f = zeros(1,events*2);
     for n = 1:events
         if n < (events)
             f(n+aux)= freqs_in_range(n)*fs/2;
             f(n+aux+1) = freqs_in_range(n+2)*fs/2;
             aux= aux+1;
         else
             f(n+aux)= freqs_in_range(1)*fs/2;
             f(n+aux+1) = freqs_in_range(end)*fs/2;
         end
     end
     xs = zeros(events,round(window_duration*fs)+1);
     aux=0;
     for n = 1: events
         xs(n,:) = make_sinusoidal_signal(f(n+aux),f(n+aux+1),...
             50,fs,window_duration,energy);
         figure;
         plot(xs(n,:));
         title(['Sinusoidal Signal ',num2str(n)])
     end
     [noise_emg,noise_input]= noise_application(x,xs,events);
end
```

3.2.1.1 min_win_energy.m

```
function [energy] = min_win_energy(windowed_signal,min_energy)
     e = zeros(1,size(windowed_signal,1));
     for n = 1 : size(windowed_signal,1)
         e(n) = norm(windowed_signal(n,:))^2;
     end
     energy = min_energy*sum(e(:))/n;
end
```

3.2.1.2 `make_sinusoidal_signal.m`

```
function [x] = make_sinusoidal_signal(max_freq,...
min_freq,number_sines,fs,duration_seg,energy,min_energy)
    t = 0 : 1 / fs : duration_seg;
    div = (max_freq-min_freq)/number_sines;
    freqs = min_freq : div : max_freq;
    amp = randi(100,1,number_sines);
    x1 = 0;
    for k = 1 : number_sines
        x1 = x1 + amp(k)*sin(2 * pi * freqs(k) * t);
    end
    X1 = fft(x1);
    X1_aux = X1(1:round(length(X1)/2));
    e = norm(X1_aux)^2;
    if e>energy
       X1 = reduce_energy(X1,energy,e);
       x = ifft(X1,'symmetric');
    else
       x = x1;
    end
end
function [X] = reduce_energy(X1,energy,e)
    while e>energy
        X1 = X1 - 0.05.*X1;
        X1_aux = X1(1:round(length(X1)/2));
        e = norm(X1_aux)^2;
    end
    X = X1;
end
```

3.2.1.3 noise_application.m

```
function [noise_signal,noise_input]= noise_application(x,noises,events)
    if size(x,2)<size(x,1)
        x = x.';
    end
    noise_signal = x;
    space_between_noises = round(length(x)/(events*2));
    aux = 0;
    noise_input = zeros(events,2);
    startpoint = randi([1 round(length(x)/events*2)]);
    for n= 1:events
        begin = n+(space_between_noises*(aux))+startpoint;
        end_ = size(noises,2)+begin;
        noise_signal(begin:end_-1) = noise_signal(begin:end_-1) + ...
            noises(n,:);
        noise_input(n,:) = [begin end_];
        aux = aux+1;
    end
end
```

3.3 Item C

As Figuras 15 e 16, demonstram o sinal após ser inserido. Observando a Figura 15 com o sinal durante todo o período de amostragem não é possível notar nenhuma diferença.

Porém ao se ampliar o sinal, Figura 16, nota-se a inserção de algum ruido no sinal original, de acordo com a analise da figura, pode-se dizer que o ruido é composto por sinais de alta frequência.

Realizando agora a comparação entre as transformadas de Fourier do sinal original, Figura 14, e do sinal com eventos, Figura 17. É possível notar coeficientes maiores na faixa de frequência dos eventos adicionados. Porém não é possível determinar quantos eventos foram adicionados, só suas frequências predominantes. Para identificar os momentos em que foram adicionados os eventos basta realizar o calculo da equação 3.1.

$$event\_time = \frac{noise\_input_i}{f_s} \tag{3.1}$$

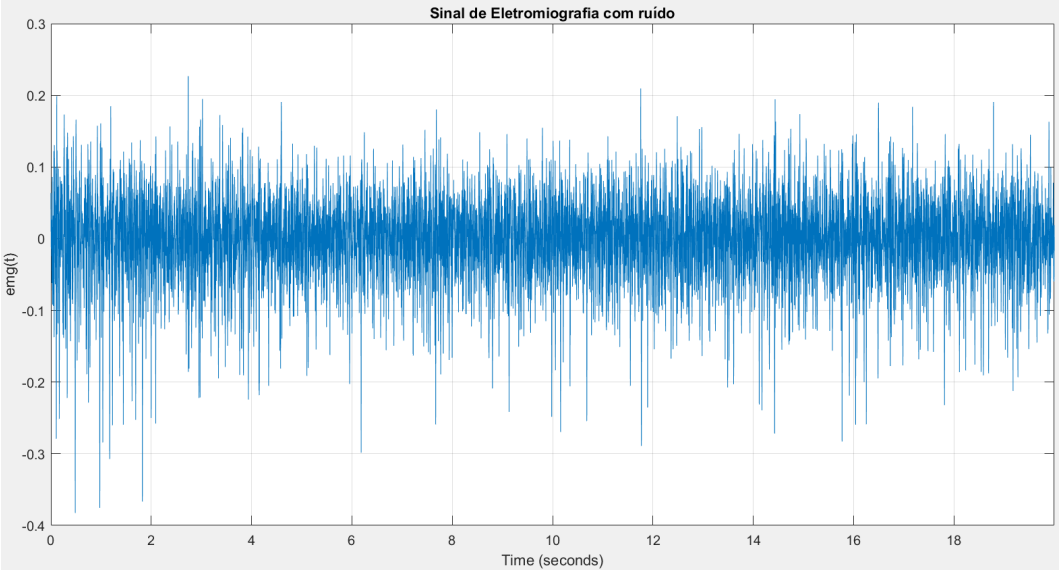
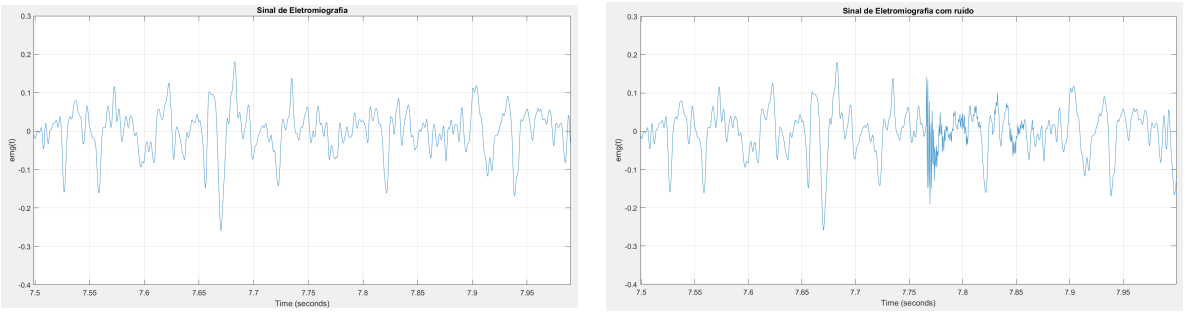


Figura 15 – Gráficos Tempo(s) x Tensão(mV) do Eletromiograma com eventos inseridos.



(a) EMG sem eventos.

(b) EMG com eventos.

Figura 16 – Comparação de faixa de tempo sem evento com faixa com evento.

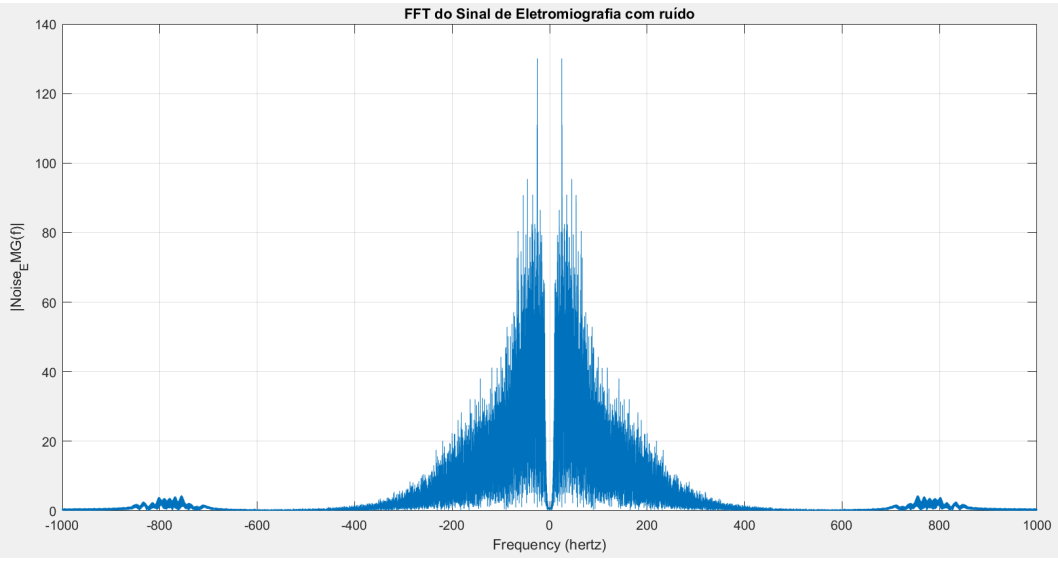


Figura 17 – Transformada de Fourier do Eletromiograma com eventos inseridos.

3.4 Item D

Para esta questão foi criado o *script* `quest3d.m`, vale lembrar que como a função `noising_signal` gera valores aleatórios, os eventos nesta questão estão alocados em locais diferentes. Os gráficos gerados pelos filtros de decomposição de Daubechie 10, *db10*, nos niveis pedidos estão nas Figuras 18, 19 e 20.

```
%quest3d.m
    load('emg1.mat','x','fs');
    min_energy=0.7;
    events = 3;
    [noise_emg,noise_input]= noising_signal(x,fs,window_duration,...
        events, 0.7,0.9,min_energy);
    [h00, h10] = wfilters('db10','d');
    for levels = 3:5
        [xd, xdc, h] = qmf_decomposition(noise_emg, h00, h10, levels);
        figure;
        plot_frequency_responses_iterated_filters(h);
        figure;
        plot(xd);
        title(['Transformada db10 do sinal de EMG, com ...
            ',num2str(levels)])
    end
    pause;close all;
    [h00, h10] = wfilters('sym8','d');
    for levels = 3:5
        [xd, xdc, h] = qmf_decomposition(noise_emg, h00, h10, levels);
        figure;
        plot_frequency_responses_iterated_filters(h);
        figure;
        plot(xd);
        title(['Transformada sym8 do sinal de EMG, com ...
            ',num2str(levels)])
    end
    pause;close all;
    [h00, h10] = wfilters('coif2','d');
    for levels = 3:5
        [xd, xdc, h] = qmf_decomposition(noise_emg, h00, h10, levels);
        figure;
        plot_frequency_responses_iterated_filters(h);
        figure;
        plot(xd);
        title(['Transformada coif2 do sinal de EMG, com ...
            ',num2str(levels)])
    end
```

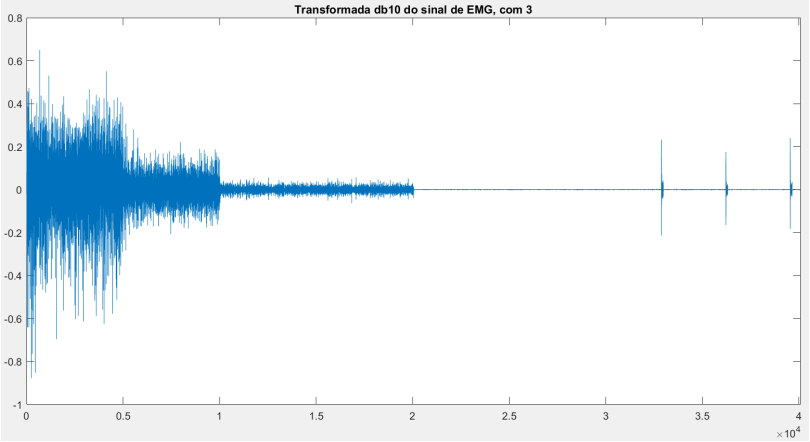


Figura 18 – 3 níveis.

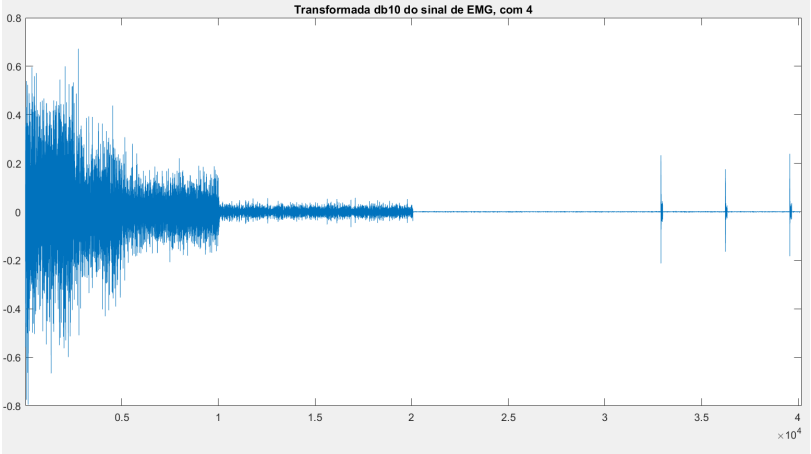


Figura 19 – 4 níveis.

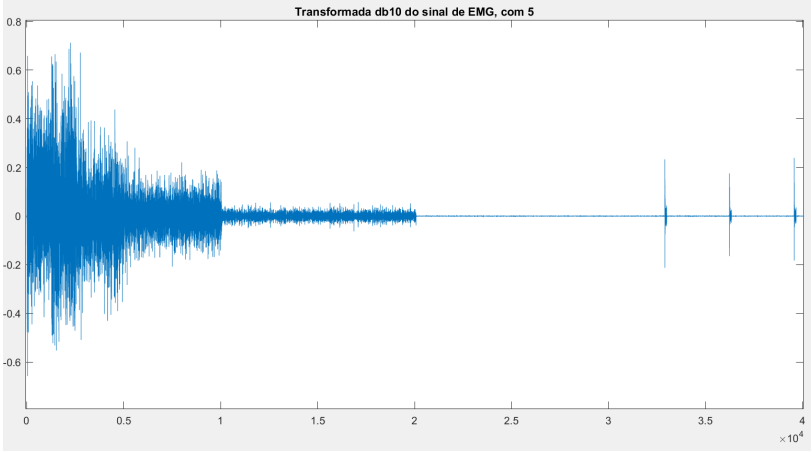


Figura 20 – 5 níveis.

3.5 Item E

Utilizou-se o script `quest3e.m`, para a realização da questão.

```
load('emg1.mat', 'x','fs');
x=x - mean(x);
wind_duration = 0.25;
t = 0 : (1/fs) : length(x)/fs;
t = t(1 : length(x));
Nfft = 10000;
levels = 4;
[h0, h1] = wfilters('db10','d');
min_energy = 0.7;
events = 3;
[noise_emg,noise_input]= noising_signal(x,fs,0.1,...
    events, 0.7,0.9,min_energy);
time_input = noise_input(:)/fs;
spect1 = signal_spec(noise_emg, fs,wind_duration,Nfft);
scalogram1 = qmf_spec(noise_emg,h0,h1,levels);
figure;
a = axes;
imagesc(0:(length(x)/fs)/Nfft:round(length(x)/fs),...
0:1:((0.5*fs*(1-1/size(scalogram1,1)))/...
(0.5*fs/size(scalogram1,1)))+1,...
flipud(scalogram1));
set(a,'YDir','normal');
xlabel('Tempo');
ylabel('Frequencia');
title('Espectrograma com base na decomposicao QMF, sinal com ...
   eventos');
caxis([min(scalogram1(:)),max(scalogram1(:))*0.1]);
colormap jet;
colorbar;
figure;
colormap jet;
imagesc(0:(length(x)/fs)/Nfft:round(length(x)/fs),...
wrev(0:0.5*fs/size(spect1,1):0.5*fs*(1-1/size(spect1,1))),spect1);
xlabel('Tempo [s]');
ylabel('Frequencia [Hz]');
title('Espectrograma com base na Transformada janelada de ...
   Fourier,sinal com eventos');
ax = gca;
ax.YDir = 'normal';
colorbar;
caxis([min(spect1(:)),max(spect1(:))*0.1]);
```

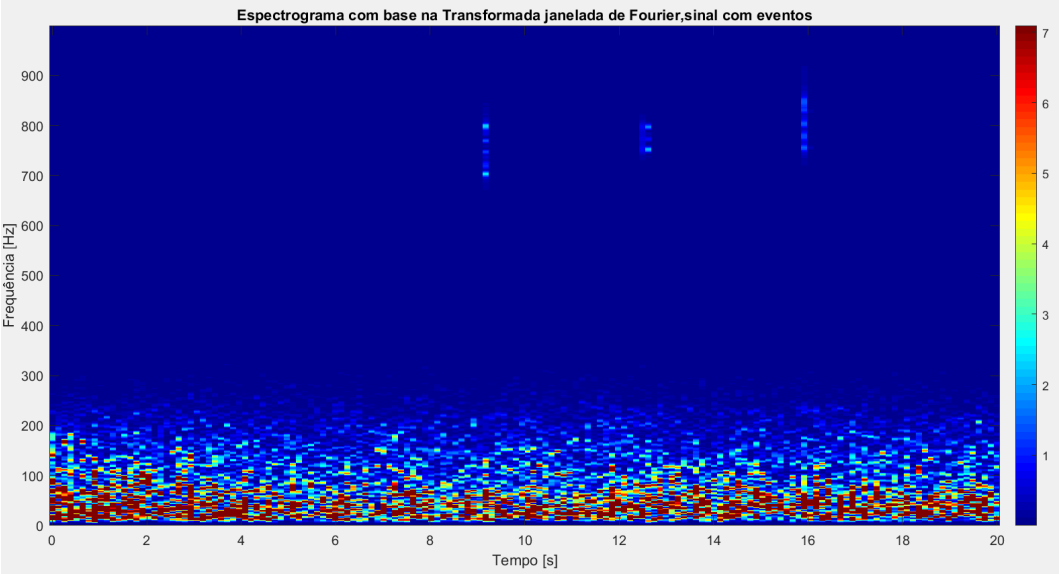


Figura 21 – Visão geral do espectrograma do sinal com eventos.

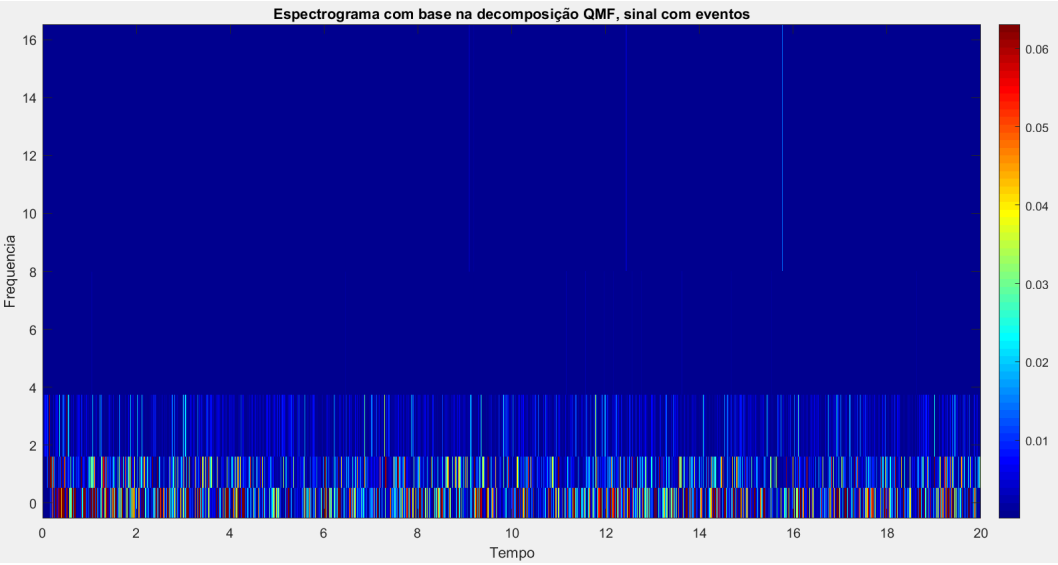


Figura 22 – Visão geral do escalograma do sinal com eventos.

O resultado do espectrograma utilizando a transformada janelada de Fourier, Figura 21, possibilita uma analise mais acurada das frequências de cada evento, porém por conta do *overlap* das janelas existe um espaçamento dos eventos no sinal ao se analisar o tempo destes não sendo exato o momento dos eventos, Figura 23.

Pela decomposição QMF é possível visualizar também a existência de 3 eventos, Figura 22, porém não é possível definir as frequências que compõe o sinal, só a escala destas, no caso são altas frequência. De forma geral é possível determinar com mais exatidão a localização temporal dos eventos, Figura 24, provando assim o afirmado na secção 1.8.

```
>> noise_input
noise_input =
        18188       18388
        24856       25056
        31524       31724
>> 18188/fs
ans = 9.0940
>> 18388/fs
ans = 9.1940
```

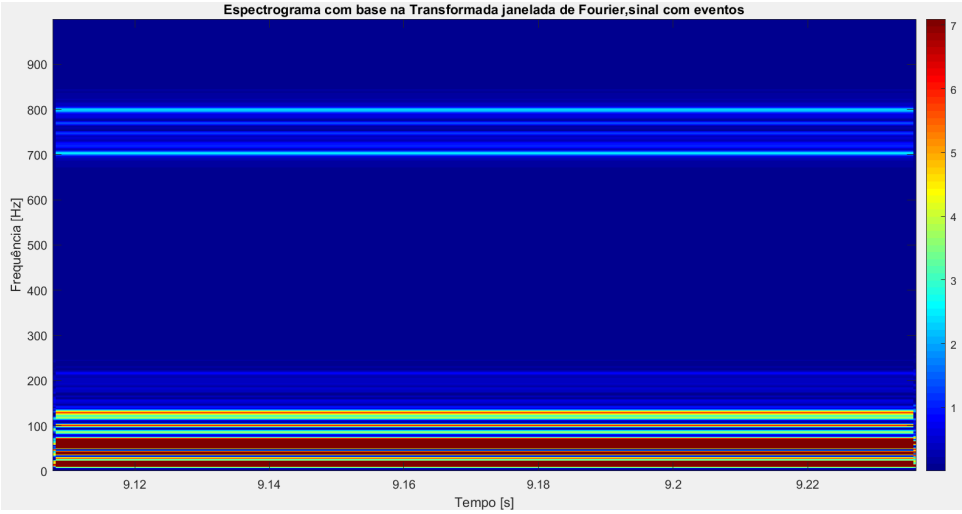


Figura 23 – Traformada janelada de Fourier, com zoom no momento do evento.

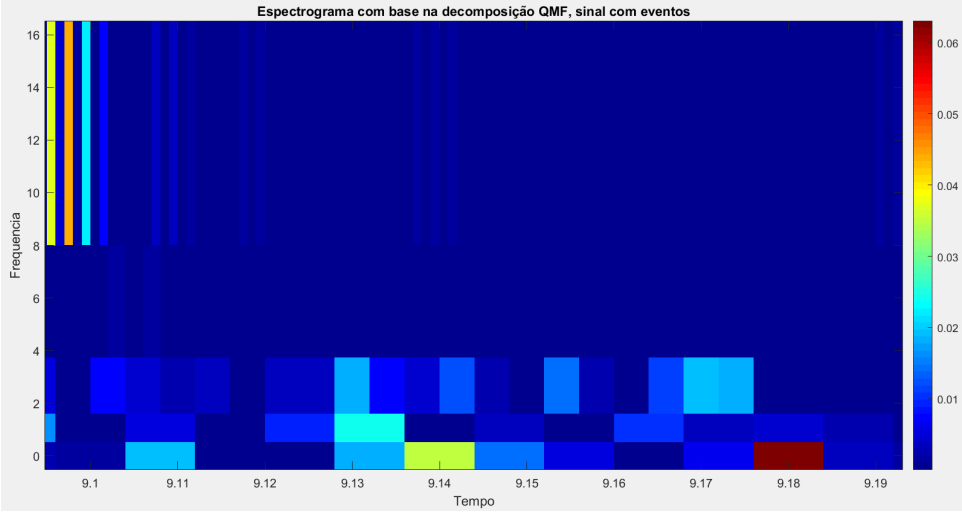


Figura 24 – Decomposição QMF, com zoom no momento do evento.

4 Questão 4

4.1 Item A

Para a realização deste item foi criada a função `NMcoefs`, os resultados foram plotados utilizando o *script* `quest4a.m` abaixo.

```
1  %quest4a.m
2  clear all; close all; clc;
3  load('../Q2/ecg_1.mat');
4  x=x-mean(x);
5  [h0,h1] = wfilters('db4','d');
6  levels = 4;
7  [x_nmcoefs, ~] = NMcoefs(x,h0,h1, levels, 5);
8  figure;plot(x_nmcoefs);
9  title('Decomposition with 5 percent of coefficients')
10 [x_nmcoefs, ~] = NMcoefs(x,h0,h1, levels, 55);
11 figure;plot(x_nmcoefs);
12 title('Decomposition with 55 percent of coefficients')
13 grid on;
14 [x_nmcoefs, ~] = NMcoefs(x,h0,h1, levels, 100);
15 figure;plot(x_nmcoefs);
16 title('Decomposition with 100 percent of coefficients')
17 grid on;
```

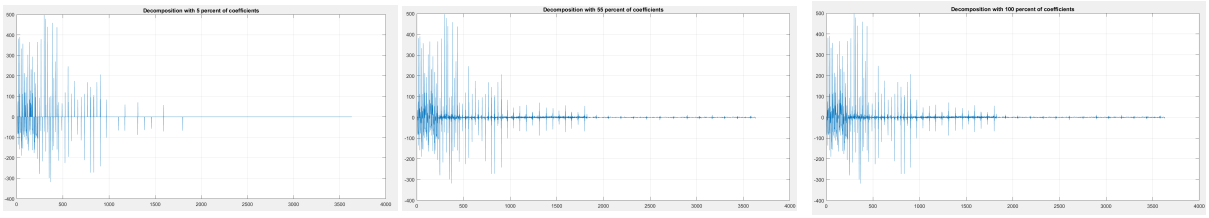
4.1.1 `NMcoefs.m`

A função realiza o calculo de quantos coeficientes representam a porcentagem inserida pelo usuário, com este valor é utilizado a função disponível no Matlab denominada `maxk` que tem com saída o numero setado como *coefs* de coeficientes máximos e os seus índices. Com isso é alocado a saída *x_nmcoefs* estes coeficientes calculados. Também é enviado como saída os tamanhos dos sinais gerados pela decomposição.

```
1 function [x_nmcoefs, lengths] = NMcoefs(x,h0,h1, levels, NM)
2     coefs = round(NM*length(x)/100);
3     [xd, xdc, ~] = qmf_decomposition(x, h0, h1, levels);
4     lengths = cellfun(@length, xdc);
5     [xd_coefs,xd_index] = maxk(xd,coefs,'ComparisonMethod','abs');
6     x_nmcoefs = zeros(1, length(xd));
7     x_nmcoefs(xd_index(1:coefs)) = xd_coefs;
8 end
```

4.1.2 Resultados

Foi realizada uma plotagem com intuito de comparação do resultado da reconstrução com diferentes porcentagens de coeficientes, Figura 25.



(a) Nm = 5%. (b) Nm = 55%. (c) Nm = 100%.

Figura 25 – Resultado da decomposição com Nm% de coeficientes.

4.2 Item B

Para a realização deste item foi criada a função `inverse_with_NMcoefs`, os resultados foram plotados utilizando o *script* `quest4b.m` abaixo.

```
1  %quest4b.m
2  clear all; close all; clc;
3  load('../Q2/ecg_1.mat');
4  x=x-mean(x);
5  [h0,h1,g0,g1] = wfilters('db4');
6  levels = 4;
7  [x_nmcoefs, lengths] = NMcoefs(x,h0,h1, levels, 5);
8  [xr] = inverse_with_NMcoefs(x_nmcoefs, lengths, h0,h1,g0,g1);
9  figure;plot(0:1/fs:((length(x)-1)/fs),xr);
10 title('Reconstruction with 5 percent of coefficients')
11 grid on;
12 [x_nmcoefs, lengths] = NMcoefs(x,h0,h1, levels, 55);
13 [xr] = inverse_with_NMcoefs(x_nmcoefs, lengths, h0,h1,g0,g1);
14 figure;plot(0:1/fs:((length(x)-1)/fs),xr);
15 title('Reconstruction with 55 percent of coefficients')
16 grid on;
17 [x_nmcoefs, lengths] = NMcoefs(x,h0,h1, levels, 100);
18 [xr] = inverse_with_NMcoefs(x_nmcoefs, lengths, h0,h1,g0,g1);
19 figure;plot(0:1/fs:((length(x)-1)/fs),xr);
20 title('Reconstruction with 100 percent of coefficients')
21 grid on;
```

4.2.1 `inverse_with_NMcoefs.m`

Os tamanhos dos sinais da decomposição que é uma saída da função da secção 4.1.1 serão utilizados para a alocação correta de cada nível. Alocados os coeficientes de cada nível basta ser utilizada a função `qmf_reconstruction`.

```
1 function [xr] = inverse_with_NMcoefs(x_nmcoefs, lengths, h0,h1,g0,g1)
2     xdc = cell(1,length(lengths));
3     for l = 1:length(lengths)
4         xdc{l} = x_nmcoefs(1:lengths(l));
5         x_nmcoefs(1:lengths(l)) = [];
6     end
7     [xr, ¬, ¬] = qmf_reconstruction(xdc, h0, h1, g0, g1);
8 end
```

4.2.2 Resultados

As Figuras 26, 27 e 28, foram geradas com intuito de demonstrar a diferença entre o sinal original e o reconstruído com as diferentes quantidades de coeficientes. Nota-se que com 5% o sinal gerado tem um encaixe alto com o sinal original, e já com 55% as diferenças são muito baixas.

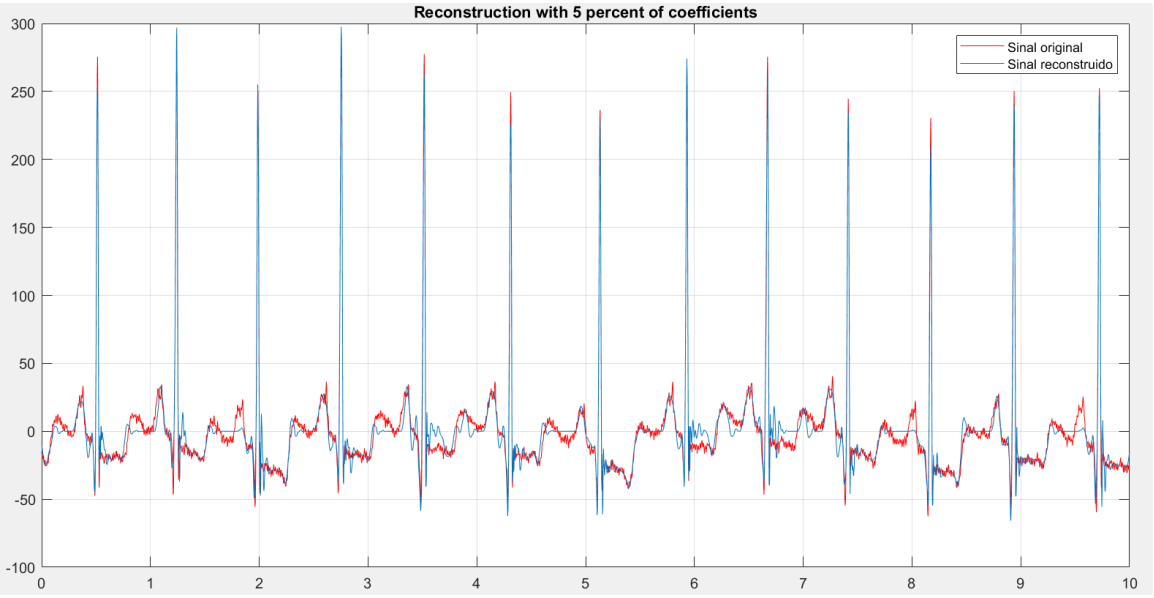


Figura 26 – Resultado da reconstrução com Nm = 5% de coeficientes.

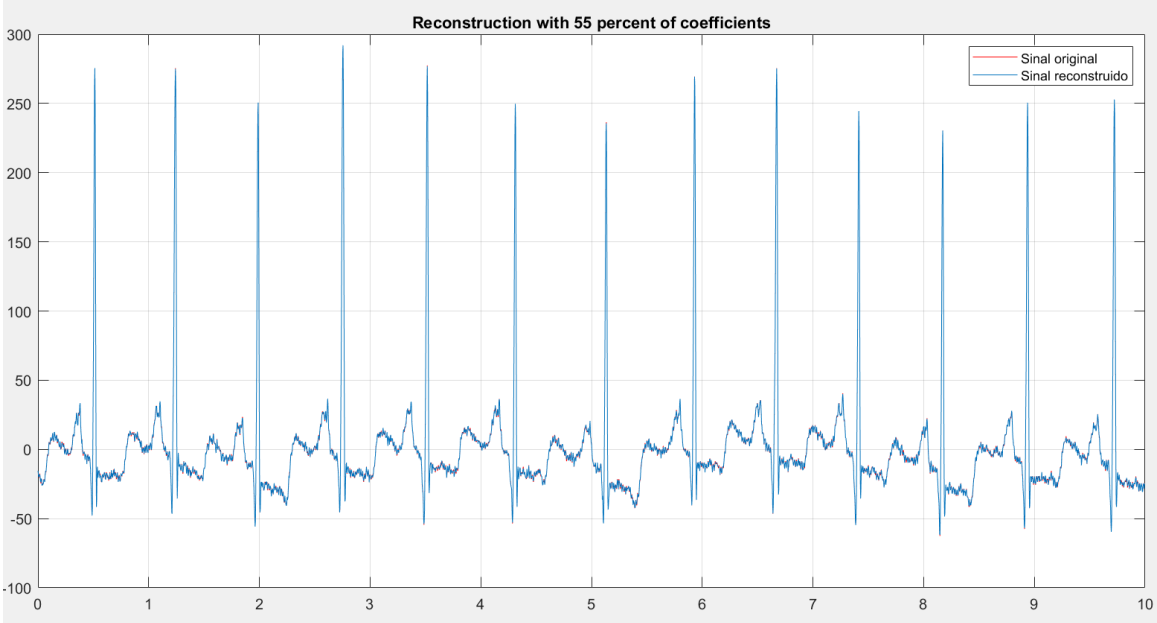


Figura 27 – Resultado da reconstrução com Nm = 55% de coeficientes.

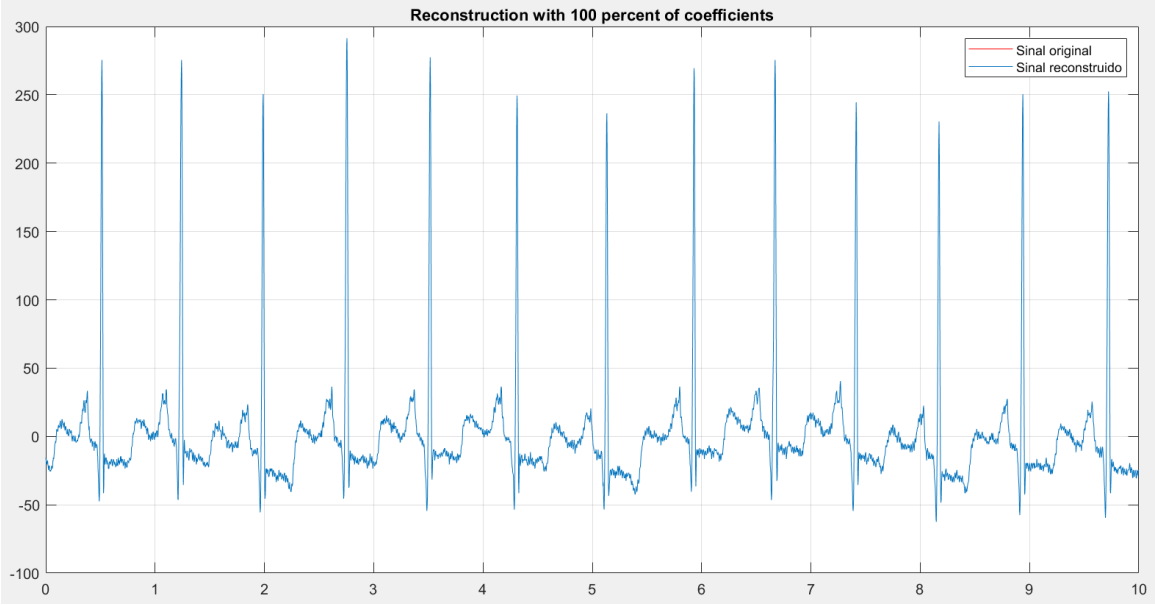


Figura 28 – Resultado da reconstrução com Nm = 100% de coeficientes.

4.3 Item C

Para esta questão foi criado o *script* `quest4c.m`, onde inicialmente foi realizado a decomposição e reconstrução com Nm de 5% à 100%, logo em seguida o calculo da relação sinal-ruído(SNR_{dB}) dos sinais reconstruídos em função do sinal original. E por fim encontrou-se o menor valor de Nm que produz uma SNR_{dB} de pelo menos 30 dB.

A Figura 29 é a curva resultante do valor da SNR_{dB} pela porcentagem de coeficientes Nm e a Figura 30 é o sinal reconstruído com a menor porcentagem que resultou no

mínimo de 30 dB. Nota-se que o sinal se parece muito com um sinal resultante de uma filtragem passa-baixa e manteve bem as características do sinal original.

```
clear all; close all; clc;
load('../Q2/ecg_1.mat');
x=x-mean(x);
NM = 5:1:100;
[h0,h1,g0,g1] = wfilters('db4');
levels = 4;
n=1;
xr = zeros(length(NM),length(x));
aux=0;
for n = min(NM):1:max(NM)
    aux = aux+1;
    [xNM, lens] = NMcoefs(x,h0,h1, levels, n);
    [xr(aux,:)] = inverse_with_NMcoefs(xNM, lens,h0,h1,g0,g1);
end
SNR = zeros(1, length(NM));
for k = 1:aux
    SNR(k) = 20*log10(norm(x)/norm(x-xr(k,:).'));
end
figure('units','normalized','outerposition',[0 0 1 1]);
plot(NM,SNR);
title('Relacao Sinal-Ruido')
xlabel('Nm coeficientes');
ylabel('SNR_{dB}')
grid on; grid minor;
aux = 0;
snr_find = 0;
while snr_find<30
    aux = aux+1;
    snr_find = SNR(aux);
end
figure('units','normalized','outerposition',[0 0 1 1]);
plot(0:1/fs:((length(x)-1)/fs),x,'r');hold on
plot(0:1/fs:((length(x)-1)/fs),xr(aux,:));
title(['Reconstruction with ', num2str(aux),' percent of coefficients'])
ylabel('mV')
xlabel('Time(s)')
legend('Sinal original','Sinal reconstruido')
grid on;
```

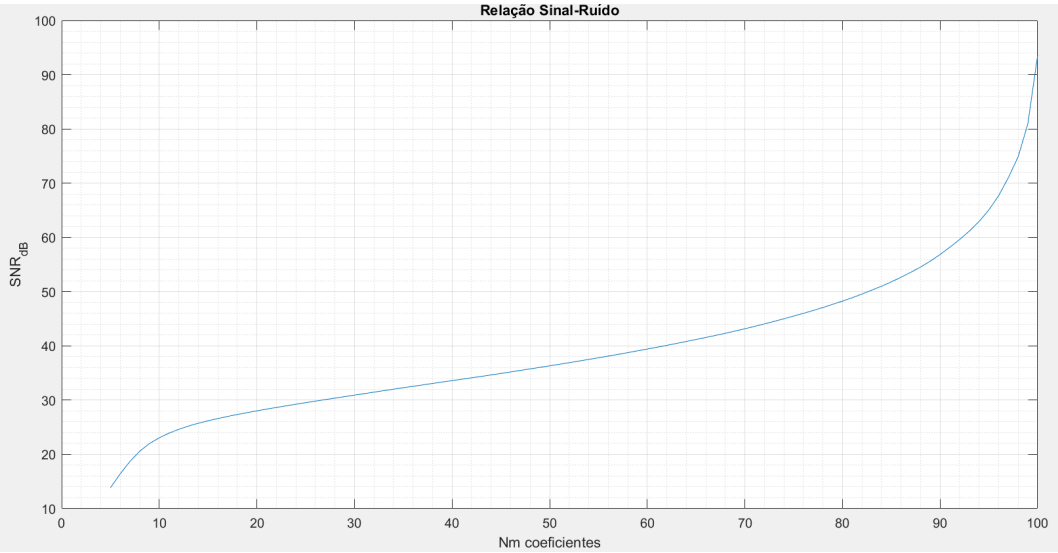


Figura 29 – Relação sinal-ruido por porcentagem de coeficientes de decomposição.

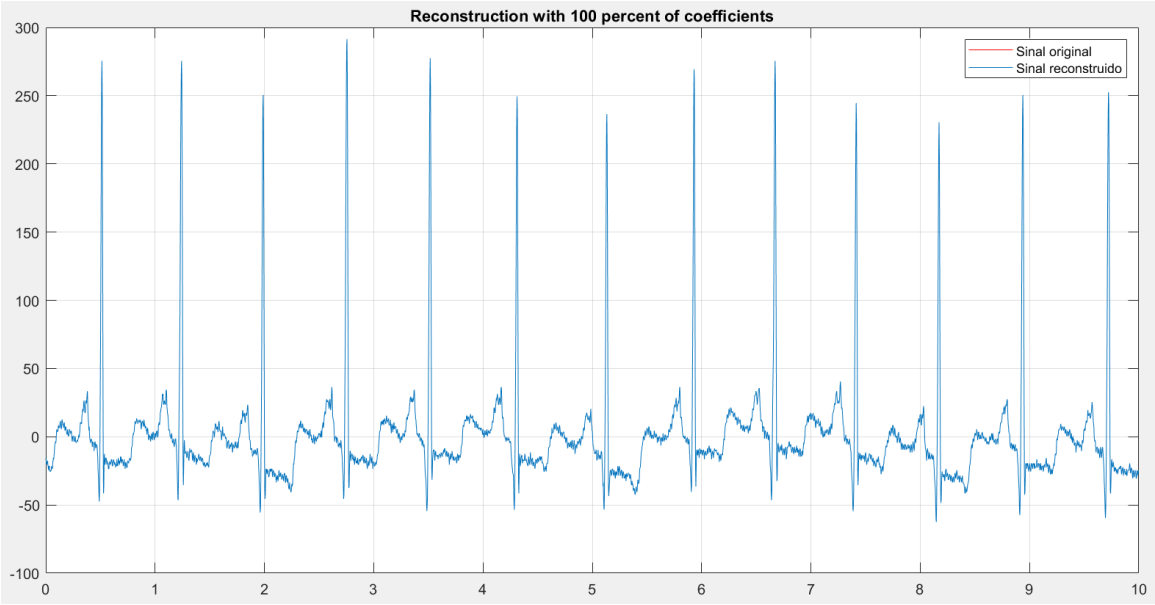


Figura 30 – Resultado da reconstrução com Nm = 23% de coeficientes.

4.4 Item D

Os principais valores para permitir a reconstrução são do complexo QRS, pois a partir destes os outros pontos são localizados. Para uma ótima reconstrução acrescentaria as ondas P e T.

5 Questão 5

5.1 extract_signals.m

```
1      function [x_a,x_b,x_c,x_d,x_e] = extract_signals();
2      D = dir('setA\*.txt');
3      for k = 1:length(D)
4          x_a{k} = dlmread(['setA/Z' sprintf('%03d', k) '.txt']);
5      end
6      D = dir('setB\*.txt');
7      for(k = 1 : length(D))
8          x_b{k} = dlmread(['setB/O' sprintf('%03d', k) '.txt']);
9      end
10     D = dir('setC\*.txt');
11     for(k = 1 : length(D))
12         x_c{k} = dlmread(['setC/N' sprintf('%03d', k) '.txt']);
13     end
14     D = dir('setD\*.txt');
15     for(k = 1 : length(D))
16         x_d{k} = dlmread(['setD/F' sprintf('%03d', k) '.txt']);
17     end
18     D = dir('setE\*.txt');
19     for(k = 1 : length(D))
20         x_e{k} = dlmread(['setE/S' sprintf('%03d', k) '.txt']);
21     end
22 end
```

5.2 Item A

Para se realizar a questão foi criado o *script* `questao5a.m` e utilizou-se as funções `carac_time_frequency` e `extract_signals`.

No *script* da questão foi realizada uma forma para visualização dos valores de todos os sinais. Os resultados encontrados para os primeiros sinais de cada grupo se encontra na Tabela 5.

Tabela 5 – Médias das características da Janelas, para os 5 grupos.

Grupos	FMediana	FMedia	FModal	VMR	RMS
A	5.4229	7.0337	2.4501	33.62	42.868
B	6.4542	7.3474	2.6752	39.075	49.048
C	2.7336	3.736	2.1751	38.295	48.418
D	1.7974	3.6314	1.4001	23.455	28.721
E	7.3273	9.026	3.6752	360.43	474.38

```
1      clear all;close all;clc
2  fs = 173.61;
3  window_duration = 5;
4  overlap = 0.5;
5  window_type = @hamming;
6  [x_a,x_b,x_c,x_d,x_e] = extract_signals();
7  M_mv = zeros(5);
8  for n = 1: 100
9    [signal_rmsA, signal_mvA, ...
         freq_modA,freq_meanA,freq_medA,carac_matrixA]...
10     = carac_time_frequency...
11     (x_a{n},fs,window_duration,window_type,overlap);
12     [signal_rmsB, signal_mvB, ...
           freq_modB,freq_meanB,freq_medB,carac_matrixB]...
13     = carac_time_frequency...
14     (x_b{n},fs,window_duration,window_type,overlap);
15     [signal_rmsC, signal_mvC, ...
           freq_modC,freq_meanC,freq_medC,carac_matrixC]...
16     = carac_time_frequency...
17     (x_c{n},fs,window_duration,window_type,overlap);
18     [signal_rmsD, signal_mvD, ...
           freq_modD,freq_meanD,freq_medD,carac_matrixD]...
19     = carac_time_frequency...
20     (x_d{n},fs,window_duration,window_type,overlap);
21     [signal_rmsE, signal_mvE, ...
           freq_modE,freq_meanE,freq_medE,carac_matrixE]...
22     = carac_time_frequency...
23     (x_e{n},fs,window_duration,window_type,overlap);
24     M_mv = [mean(freq_medA) mean(freq_meanA) mean(freq_modA)...
25     mean(signal_mvA) mean(signal_rmsA);
26     mean(freq_medB) mean(freq_meanB) mean(freq_modB)...
27     mean(signal_mvB) mean(signal_rmsB);
28     mean(freq_medC) mean(freq_meanC) mean(freq_modC)...
29     mean(signal_mvC) mean(signal_rmsC);
30     mean(freq_medD) mean(freq_meanD) mean(freq_modD)...
31     mean(signal_mvD) mean(signal_rmsD);
32     mean(freq_medE) mean(freq_meanE) mean(freq_modE)...
33     mean(signal_mvE) mean(signal_rmsE)];
34     tabelaMV = array2table(M_mv,...
35     'VariableNames',{'FMediana','FMedia','FModal','VMR','RMS'});
36     tabelaMV.Properties.RowNames = {'A','B','C','D','E'};
37     tabelaMV
38     pause; close all;
39 end
```

5.2.1 carac_time_frequency.m

Para a extração das características no domínio do tempo, o sinal foi janelado com uma janela retangular, assim não havendo alterações das suas características neste domínio, após isso as características pedidas foram extraídas com funções do próprio MATLAB.

Já para a frequência modal, o sinal foi janelado de acordo com a janela setada pelo usuário, após foi realizada a FFT do sinal janelado e retirando-se as frequências negativas, logo após foi necessário a conversão para a frequência em Hertz, assim utilizando a função `max` foi encontrada a frequência modal normalizada.

```
1   function [signal_rms, signal_mv, ...
        freq_mod,freq_mean,freq_med,carac_matrix]...
2   = ...
        carac_time_frequency(x,fs,window_duration,window_type,overlap,n_fft)
3   N_window= round(window_duration*fs);
4   x = x-mean(x);
5   if ~exist('window_type')
6      window_type = @hamming;
7   end
8   if ~exist('n_fft')
9      n_fft = N_window;
10  end
11  if ~exist('overlap')
12      overlap = 0.5;
13  end
14   w = window_type(N_window);
15   rectan = rectwin(N_window);
16  if size(x,2) > size(x,1)
17      w = w.';
18      rectan = rectan.';
19  end
20  window_shift = round((1-overlap)*N_window);
21  begin = 1;
22  end_ = N_window;
23  n=1;
24  L = round(length(x)/window_shift);
25  signal_rms = zeros(L,1);
26  signal_mv = zeros(L,1);
27  freq_mod = zeros(L,1);
28  freq_mean = zeros(L,1);
29  freq_med = zeros(L,1);
30  while end_<length(x)
31      time_signal = x(begin:end_).*rectan;
32      signal_rms(n,1) = rms(time_signal);
```

```
33          signal_mv(n,1) = mean(abs(time_signal));
34          w_signal_aux = x(begin:end_).*w;
35          W_signal_aux = fft(w_signal_aux,n_fft);
36          W_signal = W_signal_aux(1:floor(n_fft/2));
37          WX = abs(W_signal).^2;
38          [¬,freq] = max(WX);
39          freq_mod(n,1) = freq*fs/(length(WX)*2);
40          freq_mean(n,1) = meanfreq(time_signal,fs);
41          freq_med(n,1) = medfreq(time_signal,fs);
42          begin = begin + window_shift;
43          end_ = end_ + window_shift;
44          n= n + 1;
45      end
46      signal_rms =  signal_rms(1:n-1,:);
47      signal_mv  =  signal_mv(1:n-1,:);
48      freq_mod   =  freq_mod(1:n-1,:);
49      freq_mean  =  freq_mean(1:n-1,:);
50      freq_med   =  freq_med(1:n-1,:);
51      carac_matrix = [signal_rms signal_mv freq_mod freq_mean...
52          freq_med];
53  end
```

5.3 Item B

O *script* para esta questão é bem parecido com o `quest5a.m` por tanto não será explicitado. A função da secção 5.2.1, recebeu algumas alterações para extrair as proporções de energia de cada tipo de onda, será explicitado a diferença a partir da linha 30 da função, secção 5.4.

A função `energy_bands` foi inserida para extração das energias para cada cada banda característica das ondas.Os resultados encontrados para os primeiros sinais de cada grupo se encontra na Tabela 6.

Tabela 6 – Médias das porcentagens de energia relacionada a cada onda por janela.

Grupos	DELTA(δ)	TETA(θ)	ALFA(α)	BETA(β)	GAMA(γ)
A	45.533	15.548	32.278	7.7824	0.51003
B	43.334	9.4948	41.685	6.0285	0.72524
C	70.215	22.386	9.3734	0.9714	0.12033
D	79.229	12.487	8.8922	2.1686	0.49982
E	34.597	16.671	32.05	13.873	0.30806

5.4 carac_time_frequency_energy.m

```
    ...
    E = zeros(L,5);
    Epercent = zeros(L,5);
    while end_<length(x)
        time_signal = x(begin:end_).*rectan;
        signal_rms(n,1) = rms(time_signal);
        signal_mv(n,1) = mean(abs(time_signal));
        w_signal_aux = x(begin:end_).*w;
        W_signal_aux = fft(w_signal_aux,n_fft);
        W_signal = W_signal_aux(1:floor(n_fft/2));
        WX = (W_signal);
        [¬,freq] = max(WX);
        freq_mod(n,1) = freq*fs/(length(WX)*2);
        freq_mean(n,1) = meanfreq(x(begin:end_),fs);
        freq_med(n,1) = medfreq(x(begin:end_),fs);
        begin = begin + window_shift;
        end_ = end_ + window_shift;
        f = linspace(0, 0.5, length(WX))*fs;
        if n == 1
            [E(1,:), num0, den0] = energy_bands(w_signal_aux,fs);
        else
            [E(n,:), num0, den0] = energy_bands(w_signal_aux,fs,...
                num0, den0);
        end
        n= n + 1;
    end
    signal_rms =  signal_rms(1:n-1,:);
    signal_mv  =  signal_mv(1:n-1,:);
    freq_mod   =  freq_mod(1:n-1,:);
    freq_mean  =  freq_mean(1:n-1,:);
    freq_med   =  freq_med(1:n-1,:);
    E          =  E(1:n-1,:);
    Epercent   =  Epercent(1:n-1,:);
    carac_matrix = [signal_rms signal_mv freq_mod freq_mean...
        freq_med E];
end
```

5.4.1 energy_bands.m

Função que adquire as porcentagens de energias para as bandas das ondas características do EEG em relação a janela.

```
1     function [E, num, den] = energy_bands(x, fs, num0, den0)
2     maximun_frequency_hertz = fs/2;
3     number_bands = 5;
4     filter_tol = 1e-5;
5     iir_type = @butter;
6     maximum_order = 15;
7     maximun_normalized_frequency = maximun_frequency_hertz/fs;
8     bands = [0 4/fs; 4/fs 7/fs;7/fs 15/fs;16/fs 31/fs;32/fs ...
          maximun_normalized_frequency];
9     Ewindow = norm(x)^2;
10    E = zeros(1,number_bands);
11    if (¬exist('num0')) || (¬exist('den0'))
12        num{number_bands} = [];
13        den{number_bands} = [];
14        for b = 1 : number_bands
15            [num{b}, den{b}] = ...
16            iir_design(iir_type, maximum_order, ...
17            filter_tol, bands(b, :));
18        end
19    else
20        num = num0;
21        den = den0;
22    end
23    for b = 1 : number_bands
24        x_ = filter(num{b}, den{b}, x);
25        E(1, b) = 100*(norm(x_) ^ 2)/Ewindow;
26    end
27 end
```

5.5 iir_design.m

Construção do filtro IIR, identificando se com a ordem atual o filtro é estável.

```
1     function [num, den] = iir_design(iir_type, maximum_order, ...
          filter_tol, bands)
2     not_finished = 1;
3     order = maximum_order;
4     bands_ = bands;
5     mode = 'bandpass';
6     if bands(1) == 0
7         bands_ = bands(2);
```

```
8           mode = 'low';
9       end
10      if bands(2) == 0.5
11          bands_ = bands(1);
12          mode = 'high';
13      end
14      while not_finished
15          [num, den] = iir_type(order, bands_ * 2, mode);
16          not_finished = any(abs(roots(den)) > 1 - filter_tol);
17          order = order - 1;
18      end
19  end
```

5.6 Item C

Utilizou-se uma função só para o calculo das porcentagens de energias das 10 bandas por janela, a função **percentage_extraction_band**, esta realiza o calculo do valor de f_m e chama a função **energy_in_bands** que realiza o calculo das energia com as bandas espaçadas até uma máxima frequência entregue pelo usuário.

A Figura 31 demonstra os resultados para os primeiros sinais de cada grupo.

	Banda1	Banda2	Banda3	Banda4	Banda5	Banda6	Banda7	Banda8	Banda9	Banda10
A	19.522	12.386	11.421	6.743	10.786	15.424	3.1938	1.696	1.2348	1.4552
B	17.248	13.228	7.0976	5.1459	4.973	22.702	17.59	4.291	1.3986	1.0322
C	29.328	17.394	11.665	11.664	5.5142	7.3367	4.9512	0.71671	2.3065	1.0308
D	60.152	9.8949	7.2998	2.666	3.9089	2.2483	1.527	1.3474	0.65493	0.75818
E	5.2813	24.753	12.279	5.3455	8.996	7.1559	9.5311	10.002	5.1448	3.2274

Figura 31 – Tabela com porcentagem de energia em cada banda de acordo com o Grupo.

5.7 percentage_extraction_band.m

```
1  function[E_bands_proportion] = percentage_extraction_band(x,fs,...
2      percent,number_bands,window_duration,overlap)
3      n=0;
4      percentage_power = 0;
5      while percentage_power≤percent
6          n = n+1;
7          maxF_a = (length(x)-1)*fs/(2*length(x));
8          power_band = bandpower(x,fs,[0 n]);
9          total_power = bandpower(x,fs,[0 maxF_a]);
10         percentage_power = 100*(power_band/total_power);
11     end
12     fm = n;
```

```
13      [E] = energy_in_bands(x,number_bands,window_duration,fs,fm,overlap);
14      E_bands_proportion = zeros(1,number_bands);
15      for n= 1 : number_bands
16          E_bands_proportion(1,n) =  mean(E(:,n));
17      end
18  end
```

5.7.1 energy_in_bands.m

```
1   function [E, num,den,bands] = energy_in_bands(x,number_bands, ...
        window_duration_seconds,fs,maximun_frequency_hertz,window_overlap,...
2       num0,den0,window_type,maximun_order,iir_type,filter_tol,...
3       band_overlap_factor);
4       if ¬exist('window_overlap')
5           window_overlap = 0.5;
6       end
7       if ¬exist('band_overlap_factor')
8           band_overlap_factor = 0;
9       end
10      if ¬exist('filter_tol')
11          filter_tol = 1e-5;
12      end
13      if ¬exist('iir_type')
14          iir_type = @butter;
15      end
16      if ¬exist('metod')
17          metod = 'iir';
18      end
19      if ¬exist('window_type')
20          window_type = @hamming;
21      end
22      if ¬exist ('maximun_order')
23          maximun_order  =15;
24      end
25      maximun_normalized_frequency = maximun_frequency_hertz/fs;
26      N_window =round( window_duration_seconds*fs);
27      w = window_type(N_window);
28      window_shift = round((1-window_overlap)*N_window);
29      begin = 1;
30      end_ = N_window;
31      if size(x,2)>size(x,1)
32          w= w.';
33      end
34      N = length(x);
35
36
```

```
37      bands = [(0: maximun_normalized_frequency / number_bands :...
38          maximun_normalized_frequency * (1 - 1/number_bands)).' ...
39      (maximun_normalized_frequency / number_bands :...
40      maximun_normalized_frequency / number_bands :...
41      maximun_normalized_frequency).'];
42
43      bands(2 : end, 1) = bands(2 : end, 1) - ...
44      band_overlap_factor / 2 * maximun_normalized_frequency / ...
           number_bands;
45      bands(1 : end - 1, 2) = bands(1 : end - 1, 2) + ...
46      band_overlap_factor / 2 * maximun_normalized_frequency / ...
           number_bands;
47      num{number_bands} = [];
48      den{number_bands} = [];
49      if(¬exist('num0'))||(¬exist('den0'))
50          for b  = 1 : number_bands
51              [num{b},den{b}] = ...
                   iir_design(iir_type,bands(b,:),maximun_order,filter_tol);
52          end
53      else
54          num = num0;
55          den = den0;
56      end
57      w_index = 1;
58      L = round(length(x)/window_shift);
59      E = zeros(L,number_bands);
60          while end_ < N
61              xw= x(begin : end_).*w;
62              Ewindow = norm(xw)^2;
63              for b = 1 : number_bands
64                  x_ = filter(num{b},den{b}, xw);
65                  E(w_index,b) = 100*(norm(x_) ^ 2)/Ewindow;
66              end
67              w_index = w_index + 1;
68              begin = begin + window_shift;
69              end_ = end_ + window_shift;
70          end
71          E = E(1:w_index-1,:);
72  end
```

5.8 Item D

Para esse item basta não alocar na matriz o valor médio retificado da função da secção 5.2.1.

5.9 Item E

Para a realização do item foi criado o *script* `quest5e.m` e a função `extract_caracs`.

```
clear all;close all;clc
fs = 173.61;
window_duration = 5;
overlap = 0.5;
window_type = @hamming;
percent = 95;
number_bands = 10;
[x_a,¬,¬,x_d,¬] = extract_signals();
for n = 1:length(x_a)
   x_a{n} = x_a{n} - mean(x_a{n});
   x_d{n} = x_d{n} - mean(x_d{n});
end
[caracs_t_fA,caracs_EwvA,caracs_EbandsA] = extract_caracs(x_a,fs,...
    window_duration,overlap,window_type,percent,number_bands);
[caracs_t_fD,caracs_EwvD,caracs_EbandsD] = extract_caracs(x_d,fs,...
    window_duration,overlap,window_type,percent,number_bands);
training_selection_factor = 0.6;
confusion_matrixTF  = training_validation_session(caracs_t_fA, ...
    caracs_t_fD,...
    training_selection_factor);
confusion_matrixEwv = training_validation_session(caracs_EwvA, ...
    caracs_EwvD,...
    training_selection_factor);
confusion_matrixEbands = training_validation_session(caracs_EbandsA, ...
    caracs_EbandsD,...
    training_selection_factor);
```

5.9.1 `extract_caracs.m`

Esta função extrai a características dos n sinais na célula x.

```
function [caracs_t_f,caracs_Ewv,caracs_Ebands] = ...
    extract_caracs(x,fs,window_duration,...
    overlap,window_type,percent,number_bands);
    caracs_t_f = zeros(length(x),4);
    for n = 1:length(x)
       [¬,¬,¬,¬ ,carac_matrix]...
    = carac_time_frequency5D(x{n},fs,window_duration,window_type,...
    overlap);
        caracs_mean =sum(carac_matrix,1)/size(carac_matrix,1);
        caracs_t_f(n,:) = caracs_mean;
```

```
10      end
11      caracs_Ewv = zeros(length(x),5);
12      for n = 1:length(x)
13          [¬,¬,¬,¬,¬,E,¬,¬]= carac_time_frequency2_0(x{n},fs,...
14              window_duration,window_type,overlap);
15          caracs_mean =sum(E,1)/size(E,1);
16          caracs_Ewv(n,:) = caracs_mean;
17      end
18      caracs_Ebands = zeros(length(x),number_bands);
19      for n = 1:length(x)
20          [caracs_Ebands(n,:)] = percentage_extraction_band(x{n},fs,...
21              percent,number_bands,window_duration,overlap);
22      end
23  end
```

5.9.2 Resultados

Realizando o classificador SVM com as 3 estrategias nota-se em uma primeira implementação que o classificador com melhor resultado foi o que utiliza a energia das ondas características.

Tabela 7 – Matriz de confusão utilizando as energias das ondas.

	p	n
Y	37	2
N	3	38

Tabela 8 – Matriz de confusão utilizando as energias da divisão em 10 bandas.

	p	n
Y	34	3
N	6	37

Tabela 9 – Matriz de confusão utilizando as características de tempo e frequência.

	p	n
Y	34	3
N	6	37

5.10 Item F

Utilizou-se o *script* `quest5f.m`, nele é realizada as 500 sessões e as matrizes confusões resultantes são somadas, após isso foi aplicada a formula dos índices de desempenho.

```
clear all;close all;clc
fs = 173.61;
window_duration = 5;
overlap = 0.5;
window_type = @hamming;
percent = 95;
number_bands = 10;
[x_a,¬,¬,x_d,¬] = extract_signals();
for n = 1:length(x_a)
   x_a{n} = x_a{n} - mean(x_a{n});
   x_d{n} = x_d{n} - mean(x_d{n});
end
[caracs_t_fA,caracs_EwvA,caracs_EbandsA] = extract_caracs(x_a,fs,...
    window_duration,overlap,window_type,percent,number_bands);
[caracs_t_fD,caracs_EwvD,caracs_EbandsD] = extract_caracs(x_d,fs,...
    window_duration,overlap,window_type,percent,number_bands);
training_selection_factor = 0.6;
cmTF = zeros(2);
cmEwv = zeros(2);
cmEbds = zeros(2);
for n=1:1:500
    confusion_matrixTF  = training_validation_session(caracs_t_fA, ...
        caracs_t_fD,...
         training_selection_factor);
    confusion_matrixEwv = training_validation_session(caracs_EwvA, ...
        caracs_EwvD,...
         training_selection_factor);
    confusion_matrixEbands = ...
        training_validation_session(caracs_EbandsA, caracs_EbandsD,...
         training_selection_factor);
    cmTF   = cmTF + confusion_matrixTF;
    cmEwv  = cmEwv + confusion_matrixEwv;
    cmEbds = cmEbds + confusion_matrixEbands;
end
accuracyTF  = (cmTF(1,1) + cmTF(2,2))/sum(sum(cmTF));
accuracyEwv  = (cmEwv(1,1) + cmEwv(2,2))/sum(sum(cmEwv));
accuracyEbds  = (cmEbds(1,1) + cmEbds(2,2))/sum(sum(cmEbds));
recallTF    = cmTF(1,1)/(cmTF(1,1)+cmTF(2,1));
recallEwv    = cmEwv(1,1)/(cmEwv(1,1)+cmEwv(2,1));
recallEbds    = cmEbds(1,1)/(cmEbds(1,1)+cmEbds(2,1));
precisionTF = cmTF(1,1)/(cmTF(1,1)+cmTF(1,2));
precisionEwv = cmEwv(1,1)/(cmEwv(1,1)+cmEwv(1,2));
precisionEbds = cmEbds(1,1)/(cmEbds(1,1)+cmEbds(1,2));
fscoreTF    = 2/(1/precisionTF + 1/recallTF);
fscoreEwv    = 2/(1/precisionEwv + 1/recallEwv);
fscoreEbds    = 2/(1/precisionEbds + 1/recallEbds);
```

A tabela abaixo realiza a comparação dos índices de desempenho entre as estrategias.

Tabela 10 – Índices de desempenho

	Acurácia	Sensibilidade	Precisão	medida-F
E. das Ondas	94.02%	92.83%	95.09%	93.94%
E. das 10 bandas	89.97%	89.66%	90.23%	89.94%
Carac.T-F	94.79%	94.29%	95.23%	94.76%