Felipe Lima Alcântara

Processamento de sinais Biológicos

Prova 2 de Processamento de Sinai Biológicos, 02/2019.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Brasília, DF

05 de Dezembro de 2019

Sumário

1 QUESTÃO 1 . . . 3
1.1 Item A . . . 3
1.1.1 *esparsidade_detect.m* . . . 4
1.1.2 3 níveis . . . 4
1.1.3 4 níveis . . . 5
1.1.4 5 níveis . . . 6
1.1.5 Análise de resultados . . . 7
1.2 Item B . . . 8
1.2.1 `NMcoefs_2d.m` . . . 8
1.2.1.1 `matrix_reconstruction.m` . . . 9
1.2.2 Gráficos da relação sinal-ruído . . . 10
1.3 Item C . . . 11

2 QUESTÃO 2 . . . 13
2.1 Item A . . . 13
2.1.1 `qrs_detect.m` . . . 13
2.1.1.1 `R_find.m` . . . 13
2.1.1.2 `find_S.m` . . . 14
2.1.1.3 `find_Q.m` . . . 15
2.1.1.4 `dt_confer.m` . . . 16
2.2 Item B . . . 17
2.3 Item C . . . 23
2.3.1 `windowing.m` . . . 27
2.3.2 `make_cm.m` . . . 27

1 Questão 1

1.1 Item A

As imagens a serem decompostas são as mostradas na Figura 1. Utilizou-se o *script* `Q1_1.m`, nele é realizada as Transformadas de Daubechies com *ordem* = 34, Biortogonais com *ordem* = 5 e Haar. Estes foram usados por terem características que se diferem entre si, Tabela 1.

Tabela 1 – Características das transformadas *wavelets*.

	Daubechies	Haar	Biortogonais
Ortogonal	Sim	Sim	Não
Linear	Não	Sim	Sim

```
close all;clear all;clc;
load('example2.mat','x');
x6 = x{6};
x12 = x{12};
[bior_h0, bior_h1] = wfilters('bior3.5','d');
[haar_h0, haar_h1] = wfilters('haar','d');
[db_h0, db_h1] = wfilters('db34','d');
n = 0;
xd6_bior = cell(3,1);
xd12_bior = cell(3,1);
xd6_haar = cell(3,1);
xd12_haar = cell(3,1);
xd6_db = cell(3,1);
xd12_db = cell(3,1);
for levels = 3 : 5
    n = n+1;
    [xd6_bior{n},xdc1{n}] = qmf_decomposition_2d(x6, bior_h0, ...
       bior_h1, levels);
    [xd12_bior{n},xdc2{n}] = qmf_decomposition_2d(x12, bior_h0, ...
       bior_h1, levels);
    [xd6_haar{n},xdc3{n}] = qmf_decomposition_2d(x6, haar_h0, ...
       haar_h1, levels);
    [xd12_haar{n},xdc4{n}] = qmf_decomposition_2d(x12, haar_h0, ...
       haar_h1, levels);
    [xd6_db{n},xdc5{n}] = qmf_decomposition_2d(x6, db_h0, db_h1, ...
       levels);
    [xd12_db{n},xdc6{n}] = qmf_decomposition_2d(x12, db_h0, db_h1, ...
       levels);
end
```

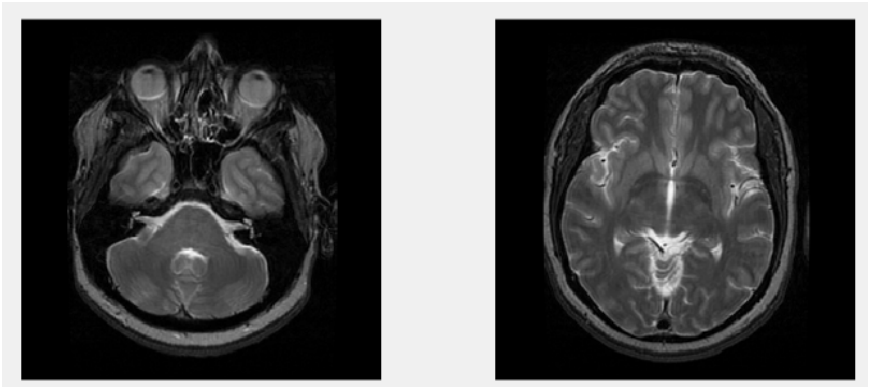

Figura 1 – Imagens de cortes axias de cabeça, obtido por RM.

Para o calculo da esparsificação foi realizada a função `esparsidade_detect`, onde buscou-se calcular a porcentagem de zeros na saída sem normalização.

1.1.1 *esparsidade_detect.m*

```
function percent = esparsidadade_detect(xdc,tol)
    if ¬exist('tol')
        tol = 1e-6;
    end
    x = 0;
    for n = 1:length(xdc)
        x = [x ;xdc{n}(:)];
    end
    x =abs(x);
    x = x/ max(x);
    [zeros] = find(x≤tol);
    percent = length(zeros)/length(x);
end
```

1.1.2 3 níveis

As Figuras 2, 3 e 4, são as respostas encontradas para decomposição de 3 níveis, para uma melhor visualização foi realizada uma normalização dos valores da imagem.

A esparsificação em porcentagem está na Tabela 2

Tabela 2 – Esparcificação com Transformada *wavelet* de três níveis.

	Daubechies	Haar	Biortogonais
X6 Zeros(%)	35.91	28.15	34.66
X12 Zeros(%)	38.17	29.72	36.46

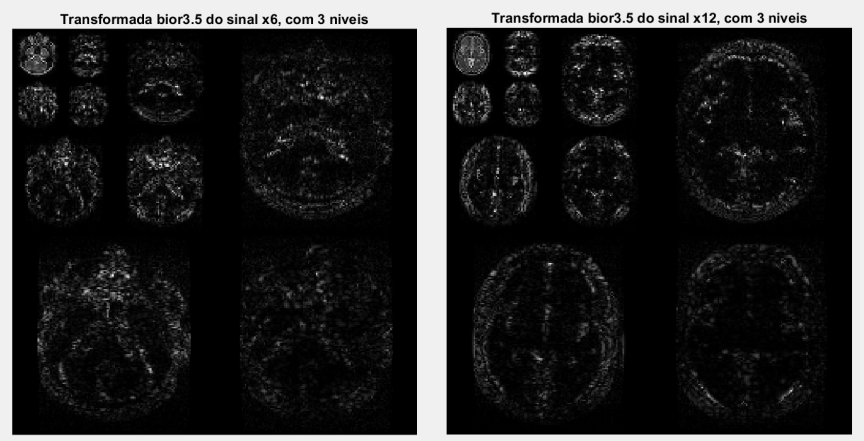

Figura 2 – Transformada *wavelet* Biortogonal, 3 níveis.

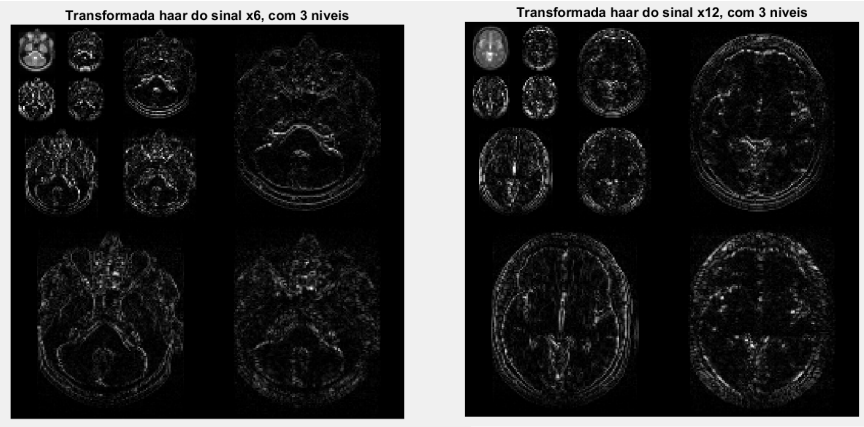

Figura 3 – Transformada *wavelet* Haar, 3 níveis.

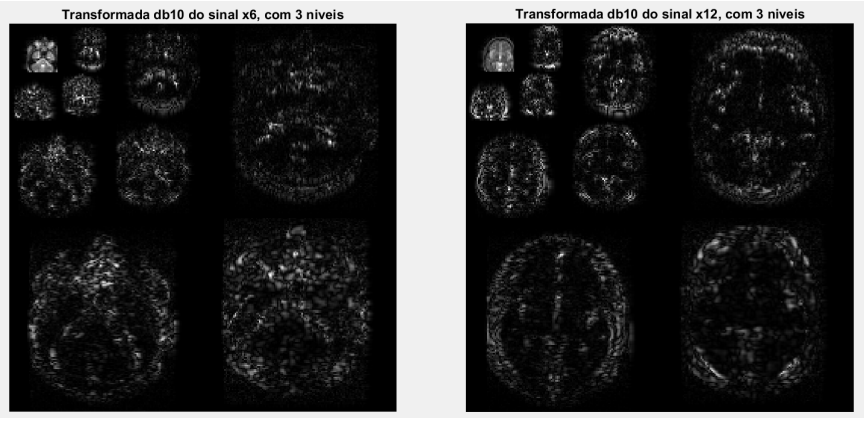

Figura 4 – Transformada *wavelet* Daubechies, 3 níveis.

1.1.3 4 níveis

As Figuras 5, 6 e 7, são as respostas encontradas para decomposição de 4 níveis. Notou-se um aumento na esparsificação, Tabela 3. As figuras estão normalizadas.

Tabela 3 – Esparcificação com Transformada *wavelet* de quatro níveis.

	Daubechies	Haar	Biortogonais
X6 Zeros(%)	37.71	28.32	35.68
X12 Zeros(%)	39.90	29.79	38.69

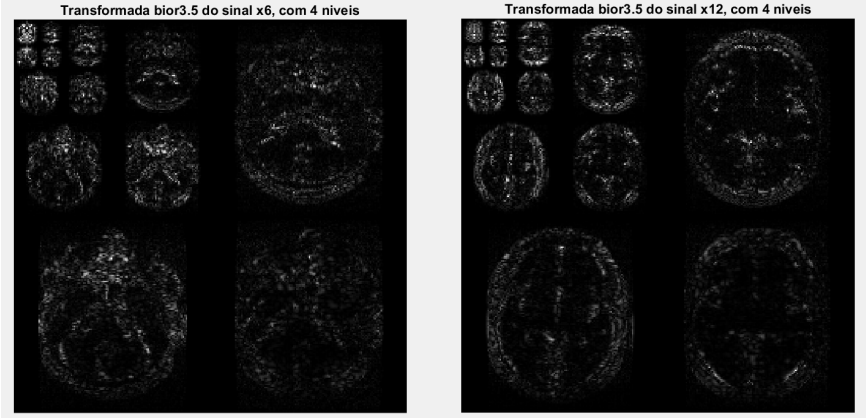

Figura 5 – Transformada *wavelet* Biortogonal, 4 níveis.

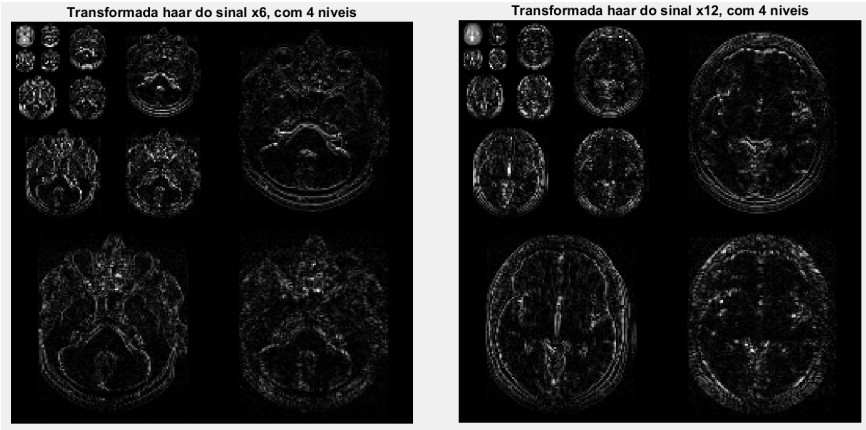

Figura 6 – Transformada *wavelet* Haar, 4 níveis.

1.1.4 5 níveis

As Figuras 8, 9 e 10, são as respostas encontradas para decomposição de 5 níveis. Novamente ouve um aumento na esparsificação da imagem, Tabela 4. As figuras estão normalizadas.

Tabela 4 – Esparcificação com Transformada *wavelet* de quatro níveis.

	Daubechies	Haar	Biortogonais
X6 Zeros(%)	39.56	28.36	37.64
X12 Zeros(%)	41.76	30.68	40.26

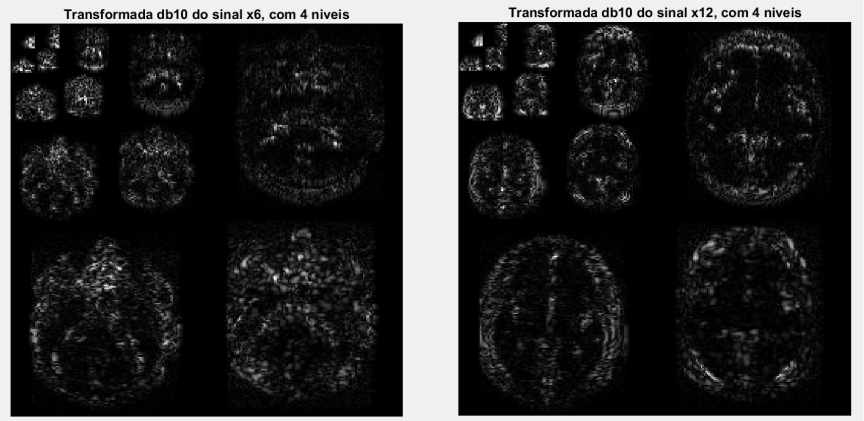

Figura 7 – Transformada *wavelet* Daubechies, 4 níveis.

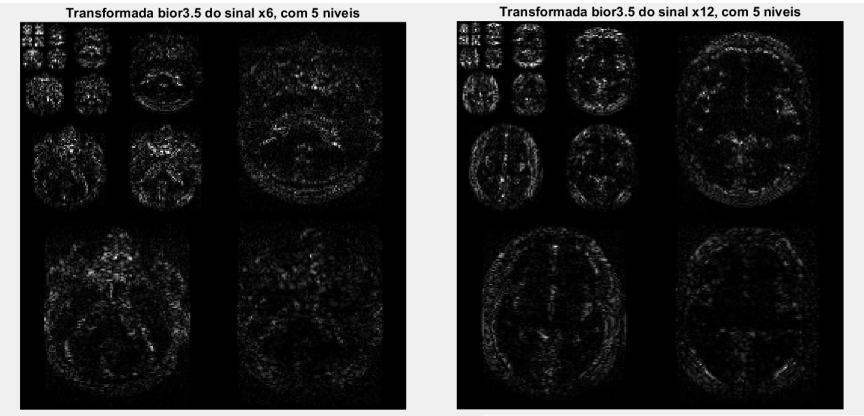

Figura 8 – Transformada *wavelet* Biortogonal, 5 níveis.

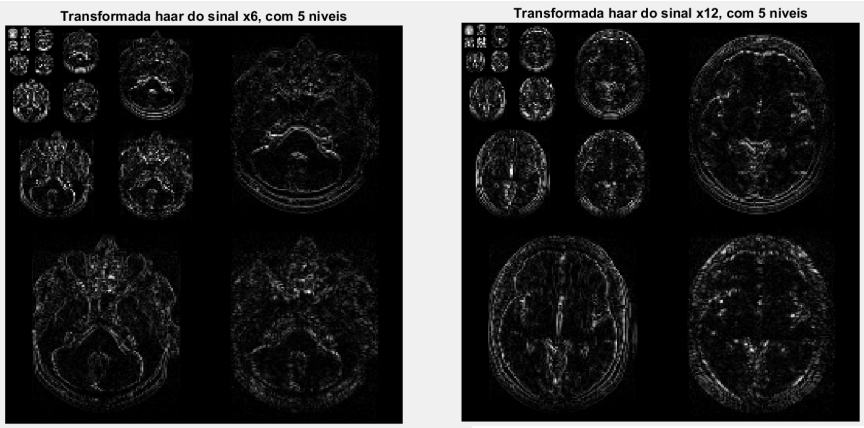

Figura 9 – Transformada *wavelet* Haar, 5 níveis.

1.1.5 Análise de resultados

A imagem que tem uma melhor resolução ao se aplicar a transformada QMF com filtros separáveis é a imagem x12, que está a direita da Figura 1. Isso já era evidente, pela imagem ter mais componentes verticais e horizontais.

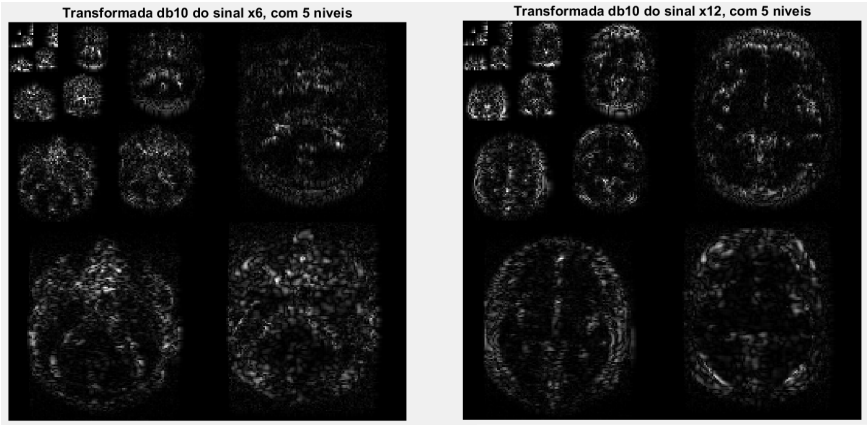

Figura 10 – Transformada *wavelet* Daubechies, 5 níveis.

1.2 Item B

O *script* `Q1_2` foi criado para realização deste item e a função `NMcoefs_2d`.

```
load('example2.mat','x');
x6 = x{6};
x12 = x{12};
NM = 5:1:100;
levels = 3;
[haar_h0, haar_h1,haar_g0,haar_g1] = wfilters('haar');
SNR_6haar3 = SNR_calc(x6,NM,haar_h0, haar_h1, haar_g0, haar_g1,levels);
SNR_12haar3 = SNR_calc(x12,NM,haar_h0, haar_h1, haar_g0, ...
    haar_g1,levels);
figure('units','normalized','outerposition',[0 0 1 1]);
plot(NM,SNR_6haar3);
hold on
plot(NM,SNR_12haar3);
legend('Trans. haar de X6','Trans. haar de X12')
title('Relacao Sinal-Ruido 3 niveis')
xlabel('Nm coeficientes');
ylabel('SNR_{dB}')
grid on; grid minor;
hold off
```

1.2.1 `NMcoefs_2d.m`

A função realiza o cálculo de quantos coeficientes da imagem 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. Após

isso as imagens são refeitas e retornadas, como *xdc_coefs*

```
function [xdc_coefs, x_nmcoefs,h,downsample_factors] = ...
    NMcoefs_2d(x,h0,h1, levels, NM,h,downsample_factors)
    if ¬exist('h') || ¬exist('downsample_factors')
        [¬, xdc, h,downsample_factors] = qmf_decomposition_2d(x, ...
            h0, h1, levels);
    else
        [¬, xdc, h] = qmf_decomposition_2d_modified(x, h0, h1, ...
            levels,h,downsample_factors);
    end
    lengths_c = cellfun(@length_c, xdc);
    lengths_r = cellfun(@length_r, xdc);
    y= [];
    for n = 1:3*levels+1
        y = [y; xdc{n}(:)];
    end
    coefs = round(NM*length(x)^2/100);
    [x_coefs,x_index] = maxk(y,coefs,'ComparisonMethod','abs');
    x_nmcoefs = zeros(1, length(y));
    x_nmcoefs(x_index(1:coefs)) = x_coefs;
    xdc_coefs = cell(1,3*levels+1);
   for n = 1:3*levels+1
       if n == 1
        [xdc_coefs{n},aux] = ...
            matrix_reconstruction(x_nmcoefs,lengths_c(n),lengths_r(n));
       else
        [xdc_coefs{n},aux] = ...
            matrix_reconstruction(aux,lengths_c(n),lengths_r(n));
       end
   end
end
function colunms = length_c(x)
    colunms = size(x,2);
end
function rows = length_r(x)
    rows = size(x,1);
end
```

1.2.1.1 `matrix_reconstruction.m`

Função para reconstrução das imagens.

```
function [matrix,x] = matrix_reconstruction(x,col,row)
    matrix = zeros(row,col);
     k = 0;
    for n = 1:col
        matrix(:,n) = x(row*k+1:row*(k+1));
        k = k+1;
    end
    x(1:row*k) = [];
end
```

1.2.2 Gráficos da relação sinal-ruído

As Figuras 11, 12 e 13, a seguir são os gráficos da relação sinal-ruído da reconstrução com NM coeficientes das Transformadas de *wavelet* com filtros de Haar, Biortogonais e Daubechies respectivamente.

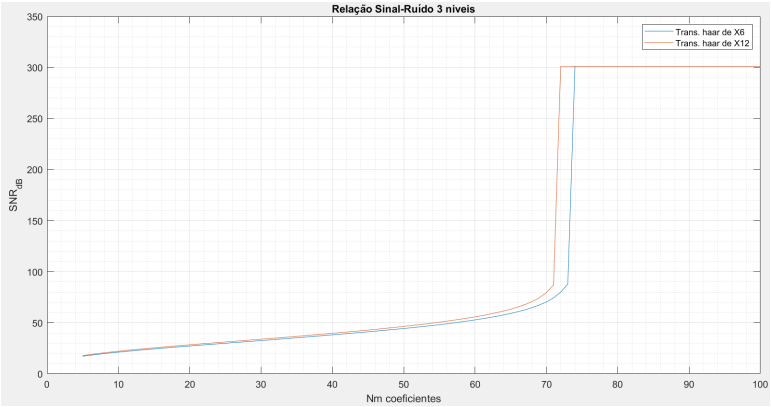

Figura 11 – SNR da reconstrução utilizando Transformada *wavelet* Haar, 3 níveis.

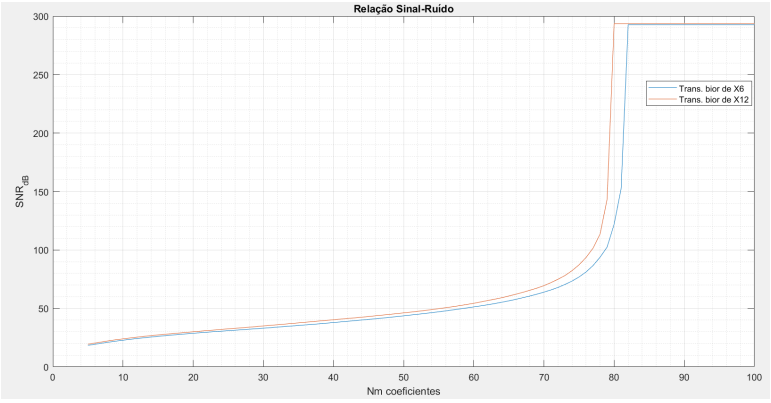

Figura 12 – SNR da reconstrução utilizando Transformada *wavelet* Biortogonal, 3 níveis.

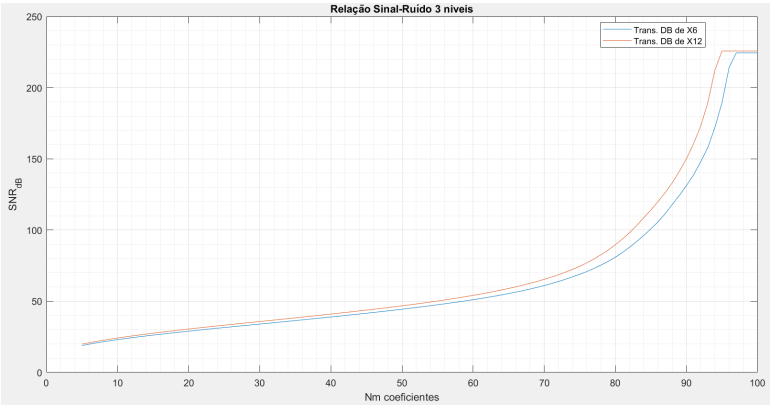

Figura 13 – SNR da reconstrução utilizando Transformada *wavelet* Daubechies, 3 níveis.

1.3 Item C

Novamente a imagem x12 tem um resultado melhor, tendo uma SNR com maior dB com o o mesmo ou menos números de coeficientes. Como nas figuras existem varias formas circulares e com bordas arredondadas, a melhor abordagem seria com filtros não separáveis.

2 Questão 2

2.1 Item A

A função criada para detecção dos picoso Q, R e S foi nomeada de `qrs_detect`. Inicialmente é retirado o *offset* do sinal, após isso é realizada uma filtragem passa banda com a banda inserida pelo usuário.

Depois inicia-se a busca pelos picos R que serviram de referência para a busca dos outros picos. A busca foi realizada buscando identificar o sinal da derivada, ao se inverter o sinal identificar o ponto de máxima para o caso do pico R e miníma para os Q e S.

2.1.1 `qrs_detect.m`

```
function [Q,R,S,Q1]= QRS_detect(x,fs,filter_bands,wind_length,M)
    filter_bands = (filter_bands/fs)*2;
    No = 2*M;
    N = No +1;
    w = hamming(N);
    [h] =  fir1(No,filter_bands,'bandpass',w);
    wx = filtfilt(h,1,x);
    [Rval,Rind] = R_find(wx,wind_length);
      R(:,1) = Rval;
      R(:,2) = Rind;
    S = zeros(size(R));
    Q = zeros(size(R));
  for n = 1 : 1 : length(Rind)
    [Sval, Spos] = find_S(Rind(n), wx,wind_length);
    S(n,1) = Sval;
    S(n,2) = Spos;
    [Qval,Qpos] = find_Q(Rind(n),wx,wind_length);
    Q(n,1) = Q1val;
    Q(n,2) = Q1pos;
  end
end
```

2.1.1.1 `R_find.m`

O ponto principal para a identificação dos picos R está que ao se filtrar o sinal, na maior parte dos casos o pico R será o máximo do sinal, assim é realizado uma comparação para só se pegar sinais que são $>= 70\% * max$, assim escolhendo-se na maioria das vezes unicamente os picos R.

```
function [value,indices] = R_find(wx,wlength)
    comp = max(wx)*0.70;
    begin = 1;
    rising = true;
    n = wlength;
    aux = 1;
    while rising
        if n ≤ length(wx)
          coefs = wx(begin:n);
          index = begin:n;
          der = diff(coefs);
          if sign(der) == 1
              n = n + 1;
              begin = begin + 1;
          else
              for k = 1: wlength
                  if coefs(k)≥ comp
                      rising = false;
                      n = n + wlength;
                      begin = begin + wlength;
                  end
              end
              if rising
                  n = n + 1;
                  begin = begin + 1;
              else
                [value(aux),ind] = max(coefs);
                indices(aux) = index(ind);
                aux = aux+1;
                rising = true;
              end
          end
        else
            rising = false;
        end
    end
end
```

2.1.1.2 `find_S.m`

O ponto central desta função está na alternância de uma derivada com sinal negativo para positivo, isto sinaliza que finalizou-se o declínio e iniciou-se a subida.

A função `dt_confer` faz a validação do fim do declínio e determina o coeficiente e

o índice do pico S.

```
function [S_val, S_pos] = find_S(Rind, x,wlength)
    n = Rind + wlength;
    begin = Rind;
    falling = true;
    while falling
       if n ≤ length(x) && begin ≥ 1
            coefs = x(begin:n);
            index = begin:n;
            der = diff(coefs);
            if sign(sum(der)) == -1
              n = n + 1;
              begin = begin + 1;
            else
              [falling,S_val,S_pos] = dt_confer(coefs,...
                  index,wlength,x,begin,n,1);
              if falling
                  n = n + 1;
                  begin = begin + 1;
              end
            end
       else
            falling = false;
            index = begin:length(x);
            [S_val,ind_aux ]= min(x(begin:end));
            S_pos = index(ind_aux);
       end
    end
end
```

2.1.1.3 `find_Q.m`

A função segue a mesma ideia que a anterior.

```
function[Qval,Qpos]=find_Q(Rind,x,wlength)
    n = Rind;falling = true;
    begin = Rind - wlength;
    if begin < 1
        begin = 1;
    end
    while falling
       if begin ≥ 1
            coefs = x(begin:n);
            index = begin:n;
```

```
11              der = diff(coefs);
12              if sign(der) == -1
13                n = n -1;
14                begin = begin -1;
15              else
16                  [falling,Qval,Qpos] = dt_confer(coefs,...
17                    index,wlength,x,begin,n,-1);
18                if falling
19                    n = n - 1;
20                    begin = begin - 1;
21                end
22              end
23          else
24              falling = false;
25              index = 1 : n;
26              [Qval,ind_aux ]= min(x(index));
27               Qpos = index(ind_aux);
28          end
29      end
30  end
```

2.1.1.4 `dt_confer.m`

Esta função tem como objetivo realizar uma validação, para que o real vale seja encontrado, pois existem pequenas elevações no sinal, que afetam a derivada e sinalizam um sinal crescente.

```
1  function [falling,valley_val,valley_pos] = dt_confer(coefs,index,...
2      wlength,x,wbegin,wend,signal)
3     if (wend+wlength)≤length(x)
4      comp = min(x(wbegin+(signal*round(wlength/2)):...
5      wend+(signal*round(wlength/2))));
6      if min(coefs)< comp
7          falling = false;
8          [valley_val,ind_aux]= min(coefs);
9          valley_pos = index(ind_aux);
10     else
11         falling = true;
12         valley_val = 0;
13         valley_pos = 0;
14     end
15    else
16        comp = min(x(wbegin+(signal*round(wlength/2)):end));
17         if min(coefs)< comp
18         falling = false;
```

```
19          [valley_val,ind_aux]= min(coefs);
20          valley_pos = index(ind_aux);
21          end
22      end
23  end
```

2.2 Item B

O primeiro sinal utilizado para teste foi o sinal *100m (0).mat*, a Figura 14 demonstra o sinal sem nenhuma alteração, o sinal filtrado durante a função `qrs_detect` está representado na Figura 15, a filtragem foi realizada com o filtro passa banda com frequência de corte 5.5 à 12 Hz, transição de 5.5 Hz com isso chega-se em uma ordem de $N_o = 64$.

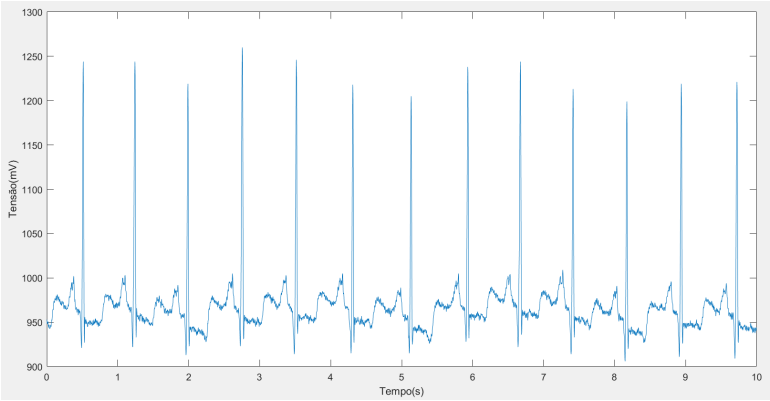

Figura 14 – Sinal de eletrocardiograma(100m(0)) no domínio do tempo.

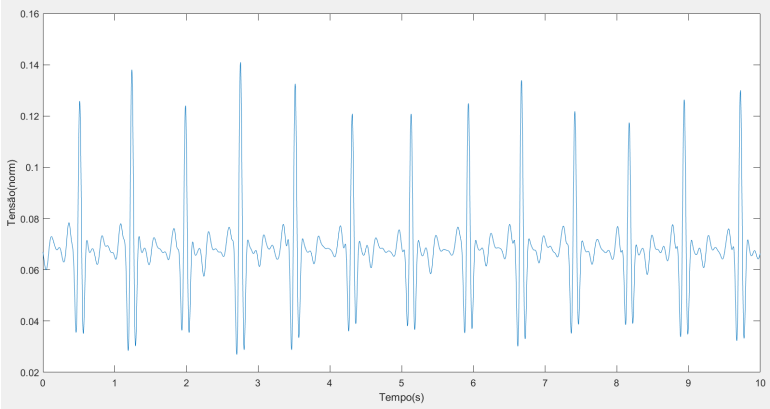

Figura 15 – Sinal resultante da filtragem, retirando-se *offset* e normalizando o sinal.

As Figuras 16 e 17 estão com marcadores nas posições onde a função detectou os picos do complexo QRS, nota-se que como o sinal utilizado para identificação é o sinal

filtrado os pontos do QRS no sinal original, marcam o complexo porém não os picos. Os acertos no sinal filtrado foram de 100%.

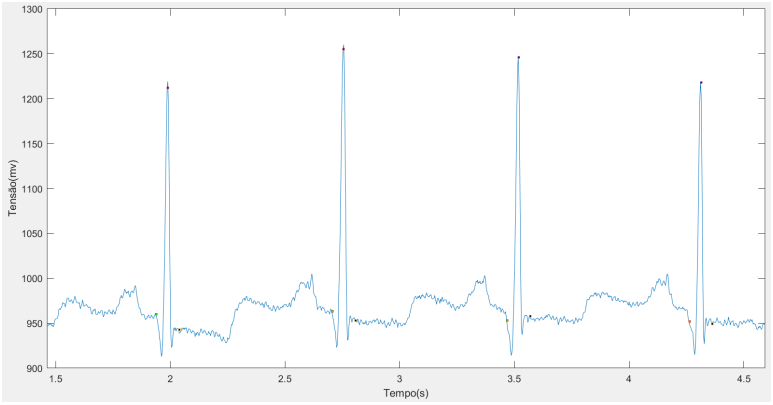

Figura 16 – Sinal de eletrocardiograma(100m(0)) no domínio do tempo com QRS marcado.

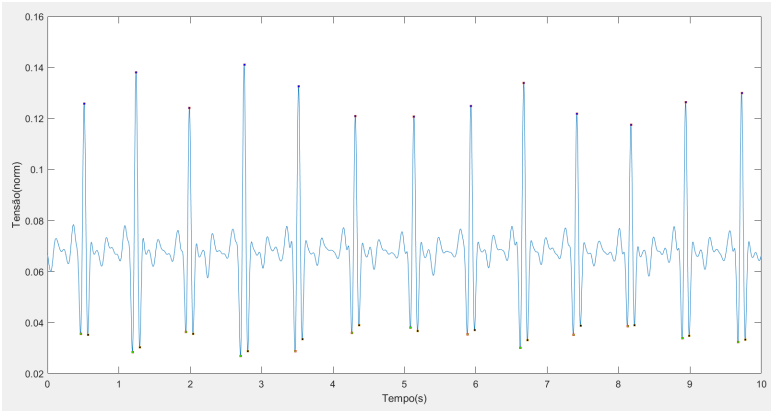

Figura 17 – Sinal filtrado com QRS marcado(100m(0)).

Em seguida realizou-se o mesmo procedimento com o sinal *114m (3)*, as Figuras 18 e 19 são as representações no domínio do tempo do sinal original e filtrado respectivamente. Nota-se que o sinal tem características diferentes do anterior sendo valido o teste.

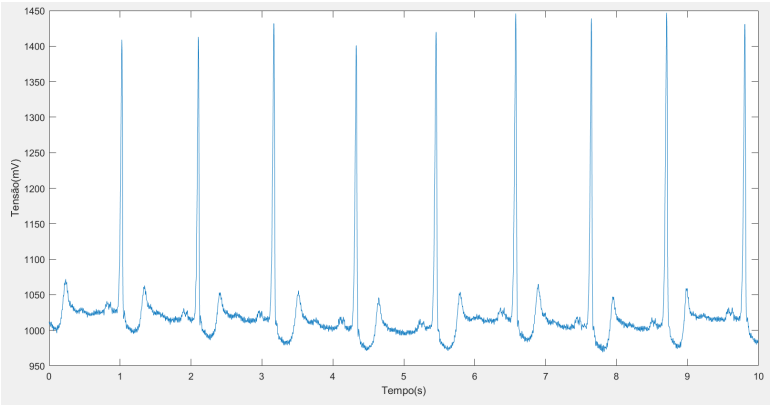

Figura 18 – Sinal de eletrocardiograma(114m(3)) no domínio do tempo.

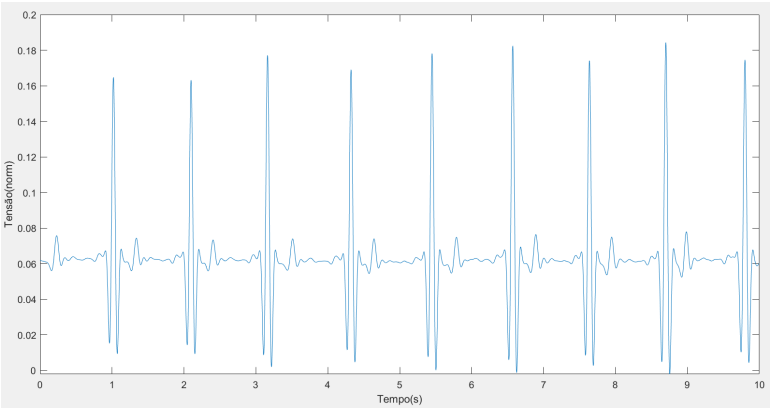

Figura 19 – Sinal resultante da filtragem, retirando-se *offset* e normalizando o sinal (114m(3)).

Plotando os marcadores, Figura 20 e 21,notou-se que com o sinal filtrado o acerto continua 100%, porém novamente no sinal original o complexo foi marcado e não os picos.

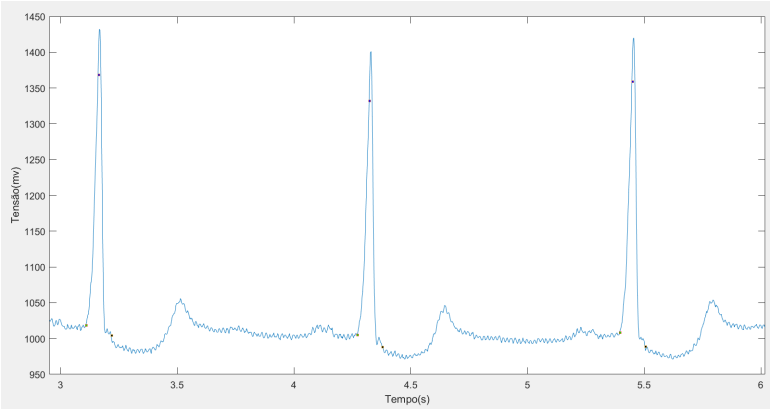

Figura 20 – Sinal de eletrocardiograma(114m(3)) no domínio do tempo.

O próximo teste foi realizado com o sinal *228m (5)*, Figuras 22 e 23, este sinal tem como característica uma proximidade grande entre o pico S e o complexo T. Os resultados

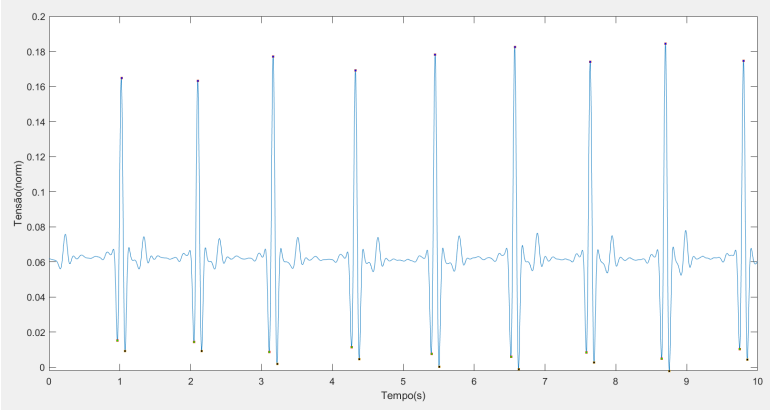

Figura 21 – Sinal resultante da filtragem, retirando-se *offset* e normalizando o sinal.

se mantiveram iguais aos anteriores.

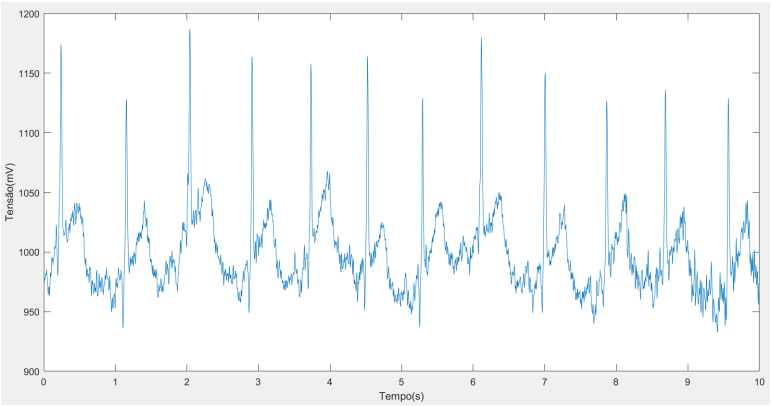

Figura 22 – Sinal de eletrocardiograma(228m(5)) no domínio do tempo.

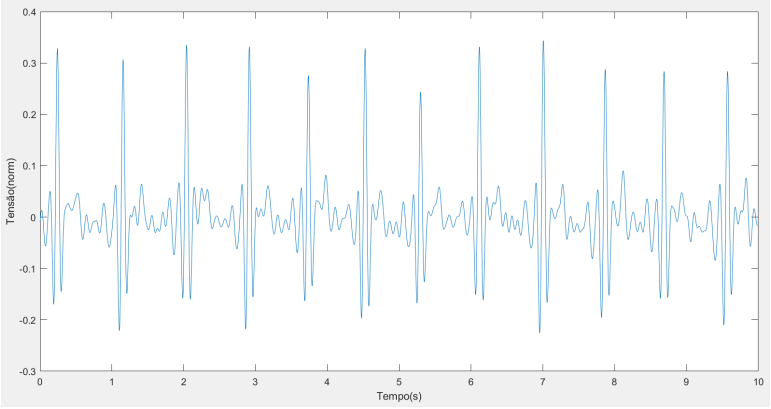

Figura 23 – Sinal resultante da filtragem, retirando-se *offset* e normalizando o sinal (228m(5)).

O sinal utilizado em seguida foi o *117m (2)*, este sinal tem o complexo T com nível bastante elevado,Figura 26, porém ocorreram não ocorreram erros, pois ao se filtrar

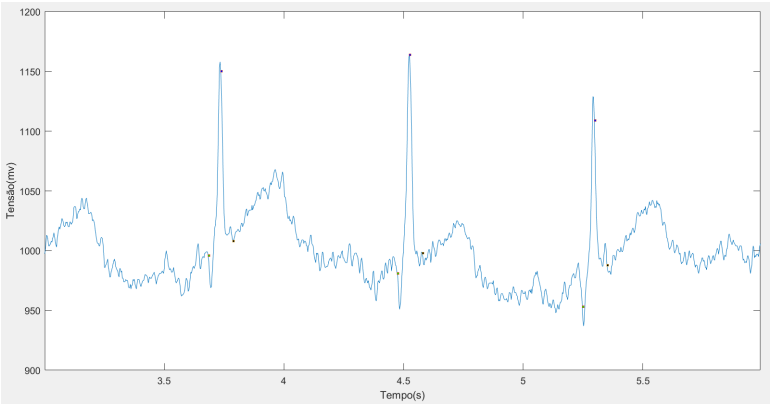

Figura 24 – Sinal original, (228m(5)), com os marcadores QRS.

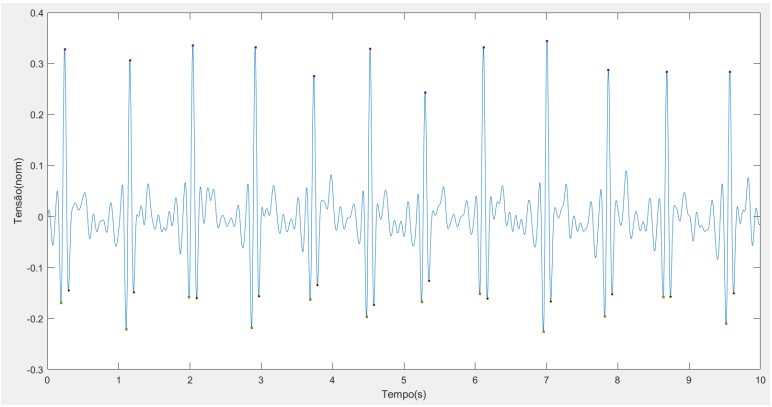

Figura 25 – Sinal resultante da filtragem do sinal 228m(5) com marcadores QRS.

o sinal o pico T diminuiu, Figura 27, e como o código utiliza uma tolerância miníma de altura dos picos, estes pontos foram excluídos, Figura 21. O resultado no sinal original foi o pior, muito distante dos picos R e também no S, Figura 28.

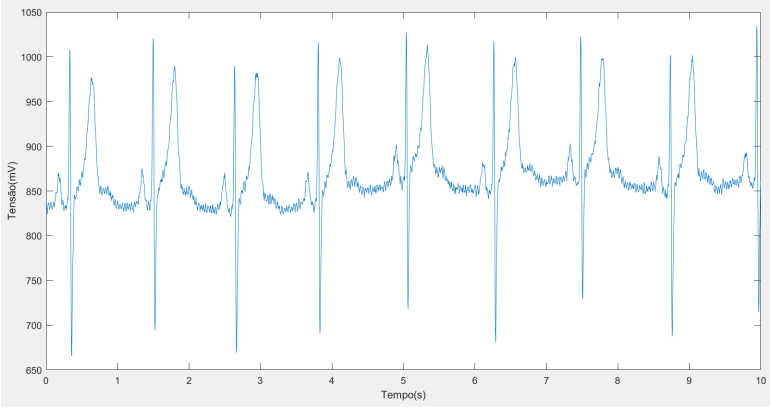

Figura 26 – Sinal de eletrocardiograma(117m(2)) no domínio do tempo.

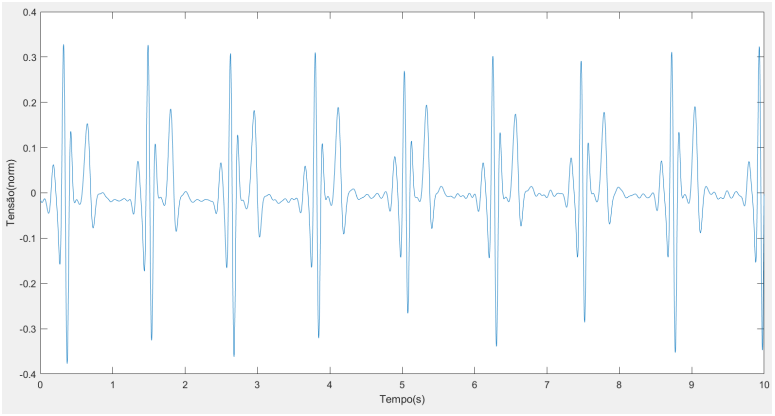

Figura 27 – Sinal resultante da filtragem, retirando-se *offset* e normalizando o sinal (117m(2)).

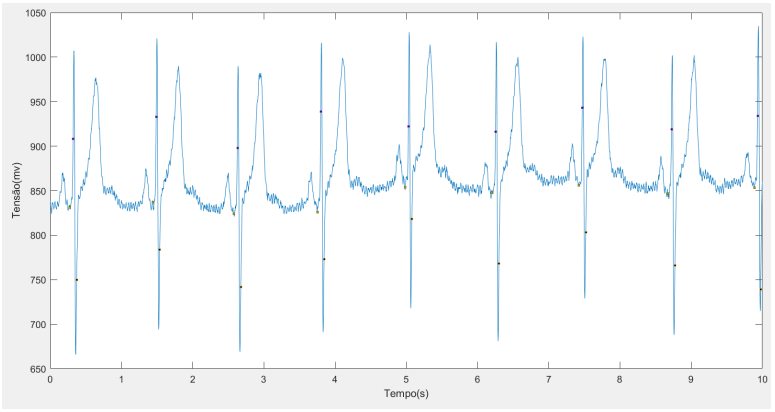

Figura 28 – Sinal original, (117m(2)), com os marcadores QRS.

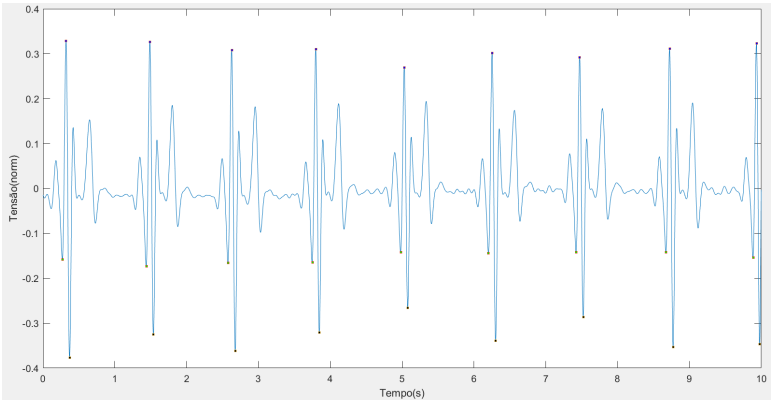

Figura 29 – Sinal resultante da filtragem do sinal 117m(2) com marcadores QRS.

Por fim foi realizado o teste com o sinal *103m (2)* um sinal com os picos Q e S pouco destacados, Figura 30. Este sinal teve o melhor resultado relacionado ao ponto de pico R, mantendo o padrão dos pontos Q e S estarem no inicio e no final do complexo respectivamente, isto para o sinal original Figura 32.

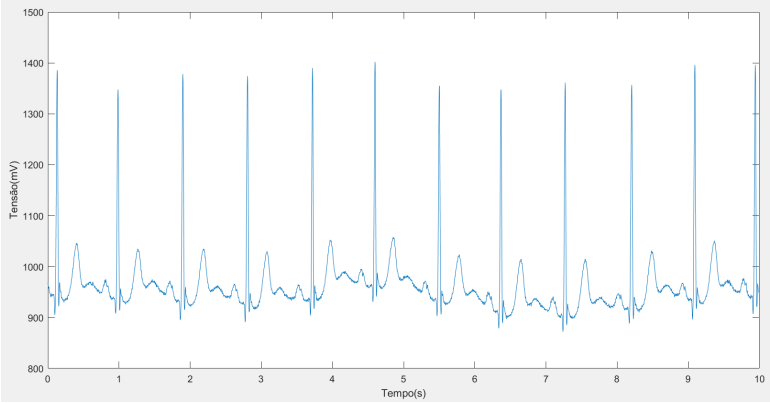

Figura 30 – Sinal de eletrocardiograma(103m(2)) no domínio do tempo.

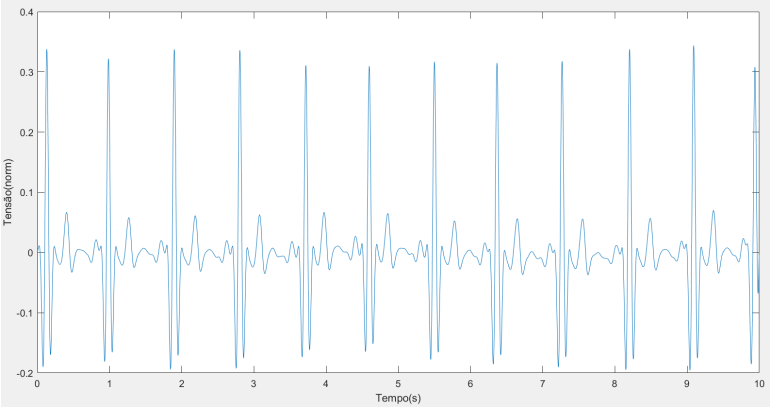

Figura 31 – Sinal resultante da filtragem, retirando-se *offset* e normalizando o sinal (103m(2)).

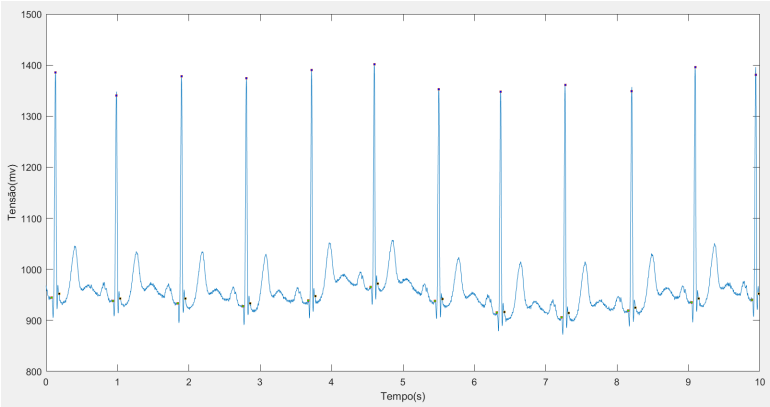

Figura 32 – Sinal original, (103m(2)), com os marcadores QRS.

2.3 Item C

Para este item foi escolhido o pico R para ser detectado e os sinais utilizados para validação e treinamento foram os sinais filtrados, alguns pontos são necessários serem explicados. Primeiramente utilizar todas as janelas cuja as quais o pico não está no centro

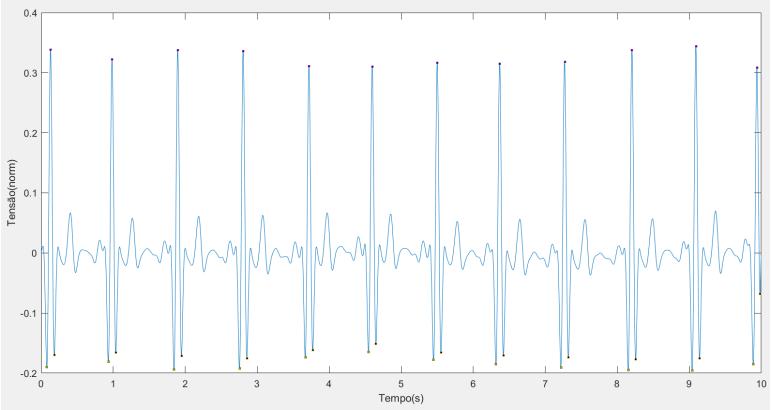

Figura 33 – Sinal resultante da filtragem do sinal 103m(2) com marcadores QRS.

mesmo com a tolerância estava desbalanceando bastante o treinamento então foi necessário balancear, assim utilizando o mesmo numero de janelas positivas e negativas.

Em segundo lugar foi diminuído o numero de sinais utilizados em 25%, adicionando a linha abaixo na função `randomly_select_training_validation`, , construída em sala. E por fim o *overlap* das janelas de treinamento foi de 0.8.

```
r1 = r(1:round(length(r)/4));
```

Realizando este balanceamento o treinamento parou de dar *overfit* e notou-se que os treinamentos funcionaram de maneira bem melhor, como pode ser analisado na matriz de confusão exemplo, Tabela 5.

Tabela 5 – Matriz de confusão utilizando `svm.predict(E_t)`.

	p	n
Y	285	0
N	9	294

Para a validação foi realizada a mudança de *overlap* para 0, não havendo sobreposição de janelas assim, da mesma forma que no treinamento, foi realizada uma entrada balanceada de janelas.

Tabela 6 – Índices de desempenho

Iteração	Acurácia	Sensibilidade	Precisão	medida-F
1^a	74.00%	84.00%	70.00%	76.36%
2^a	70.37%	88.89%	64.86%	75.00%
3^a	77.50 %	100%	68.97%	81.63%

Neste item foi criado o *script* `Q2_3.m`, que realiza a escolha dos sinais a serem utilizados para treinamento e validação com a função `randomly_select_training_validation` e a separação do sinal com a função `windowing`, uma função simples que separa o sinal em janelas retangulares com o *overlap* dado pelo usuário.

Para identificar se o pico R estava no centro, foi utilizado um vetor com as numerações das posições do sinal que também foi janelado, assim tendo os índices do sinal original que foram alocados em cada janela.

```
clear all; close all; clc;
ecg_signals_path = 'ecg/';
signalfield_name = 'val';
% Load Signals
[ecg_signals,¬,s_name] = loadmat(ecg_signals_path, signalfield_name);
for n = 1 : length(ecg_signals)
ecgs(n,:) = ecg_signals{n};
end
fs = 360;
filter_bands = ([5.5 12]/fs)*2;
No = 64;
N = No +1;
w = hamming(N);
[h] =  fir1(No,filter_bands,'bandpass',w);
window_duration = 1;
tol = 20;
wind_length = 5;
[E_training, E_validation] = ...
    randomly_select_training_validation(ecgs,0.70);
a=0;
b=0;
for i = 1: size(E_training,1)
    val = E_training(i,:);
    x = val-mean(val);
    x = x/max(abs(x));
    hx = filtfilt(h,1,x);
    [Q,R,S,Q1]= QRS_detect2(hx,wind_length);
    overlap = 0.8;
    ind = 1:3600;
    ind_janelado = windowing(ind,overlap,window_duration,fs);
    x_janelado = windowing(hx, overlap,window_duration,fs);
    for n = 1:size(ind_janelado,1)
        windows_cindex(n,:) = ind_janelado(n,...
            (size(ind_janelado,2))/2-tol:(size(ind_janelado,2))/2+tol);
        for k = 1: size(R,1)
            aux = windows_cindex(n,:) == R(k,2);
            aux = sum(aux);
            if aux > 0
                a = a+1;
                positive_sing(a,:) = x_janelado(n,:);
            else
                b = b+1;
                negative_sing(b,:) = x_janelado(n,:);
```

```
            end
        end
    end
end
E_t = [negative_sing(1:size(positive_sing,1),:);positive_sing];
t_training = [-ones(size(positive_sing,1), 1);...
ones(size(positive_sing, 1), 1)];
svm = fitcsvm(E_t, t_training, 'KernelFunction', 'linear',...
    'Standardize', true,'BoxConstraint',10);
res = svm.predict(E_t);
confusion_matrix1 = make_cm(results,t_training);
a1=0;
b1=0;
for i = 1: size(E_validation,1)
    val = E_validation(i,:);
    x = val-mean(val);
    x = x/max(abs(x));
    hx = filtfilt(h,1,x);
    [Q,R,S,Q1]= QRS_detect2(hx,wind_length);
    overlap = 0;
    ind = 1:3600;
    ind_janelado = windowing(ind,overlap,window_duration,fs);
    x_janelado = windowing(hx, overlap,window_duration,fs);
    for n = 1:size(ind_janelado,1)
        windows_cindex(n,:) = ind_janelado(n,...
            (size(ind_janelado,2))/2-tol:(size(ind_janelado,2))/2+tol);
        for k = 1: size(R,1)
            aux = windows_cindex(n,:) == R(k,2);
            aux = sum(aux);
            if aux > 0
                a1 = a1+1;
                positive_sing1(a1,:) = x_janelado(n,:);
            else
                b1 = b1+1;
                negative_sing1(b1,:) = x_janelado(n,:);
            end
        end
    end
end
E_v = [negative_sing1(1:size(positive_sing1,1),:); positive_sing1];
t_validation = [-ones(size(positive_sing1,1), 1);...
     ones(size(positive_sing1, 1), 1)];
 results = svm.predict(E_v);
confusion_matrix = make_cm(results,t_validation);
accuracy  = (confusion_matrix(1,1) ...
   +confusion_matrix(2,2))/sum(sum(confusion_matrix));
recall    = ...
```

```
      confusion_matrix(1,1)/(confusion_matrix(1,1)+confusion_matrix(2,1));
89 precision = ...
      confusion_matrix(1,1)/(confusion_matrix(1,1)+confusion_matrix(1,2));
90 fscore    = 2/(1/precisionTF + 1/recallTF);
91 desempenho = [accuracy recall precision fscore]
```

2.3.1 windowing.m

```
1  function wx = windowing(x, overlap,window_duration,fs)
2      N_window = round(window_duration * fs);
3      w = rectwin(N_window);
4      if size(x, 2) > size(x, 1)
5          w = w.';
6      end
7      window_shift = round((1-overlap)*N_window);
8      L = round(length(x)/window_shift);
9      wx = zeros(L,N_window);
10     begin = 1;
11     end_ = N_window;
12     n=1;
13     while end_<length(x)
14         wx(n,:) = x(begin:end_).*w;
15         begin = begin + window_shift;
16         end_ = end_ + window_shift;
17         n= n + 1;
18     end
19     wx = wx(1:n-1,:);
20 end
```

2.3.2 make_cm.m

```
1  function confusion_matrix = make_cm(results,t)
2   true_positives = sum((results == 1) .* (t == 1));
3   true_negatives = sum((results == -1) .* (t == -1));
4   false_positives = sum((results == 1) .* (t == -1));
5   false_negatives = sum((results == -1) .* (t == 1));
6   confusion_matrix = [true_positives false_positives; false_negatives ...
       true_negatives];
```