PROVA 2 - PROCESSAMENTO DE SINAIS BIOLÓGICOS

Davi de Alencar Mendes (dmendes@aluno.unb.br) - 16/0026415

Engenharia Eletrônica, UnB-FGA, Brasília, Brasil

1. QUESTÃO 1 - COMPACTAÇÃO DE ENERGIA EM TRANSFORMADAS WAVELET 2D

A - Decomposição separável em 3,4 e 5 níveis

Para a decomposição foram escolhidos os filtros: *db3*, *sym4* e *bior3.1*. Cada um destes apresenta propriedades diferentes em especial quanto a fase e é notório também o uso de wavelets do tipo biortogonal em aplicações de compressão de imagens com transformadas.

A eficácia de uma transformada depende da compactação de energia provida. Uma maneira de mensurar a compactação de energia é mostrada em [1], trata-se de uma medida para uma transformada ortonormal por meio da divisão entre a média aritmética da variância dos coeficientes pela sua média geométrica. Essa razão é conhecida como *Transform Coding Gain* - G_{TC}:

Incialmente, acreditoque a imagem 1 possa ser melhor representada já que qualitativamente apresenta contornos menos complexos que a imagem 2. Entretanto, a análise do G_{TC} revela que há maior ganho para a imagem 2.

$$G_{TC} = \frac{\frac{1}{N}\sum_{i=0}^{N-1}\sigma_i^2}{(\prod_{i=0}^{N-1}\sigma_i)^{1/N}} \tag{1}$$

As figuras a seguir mostram as decomposições obtidas para a imagem 1 (corte com olhos) e imagem 2 (corte cerebral).

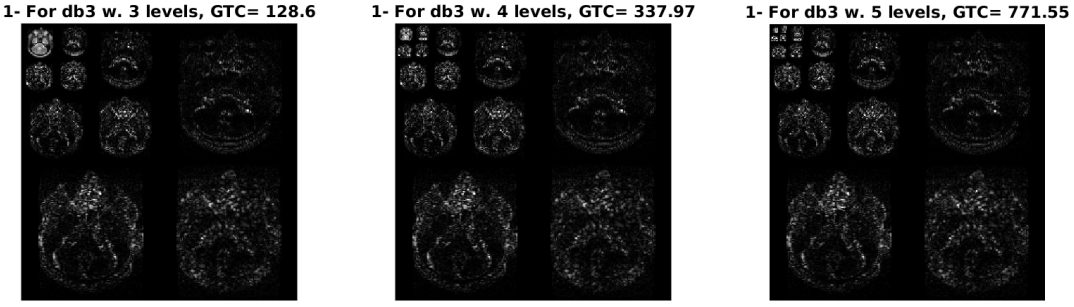

Fig. 1. Decomposição usando db3 para a imagem 1

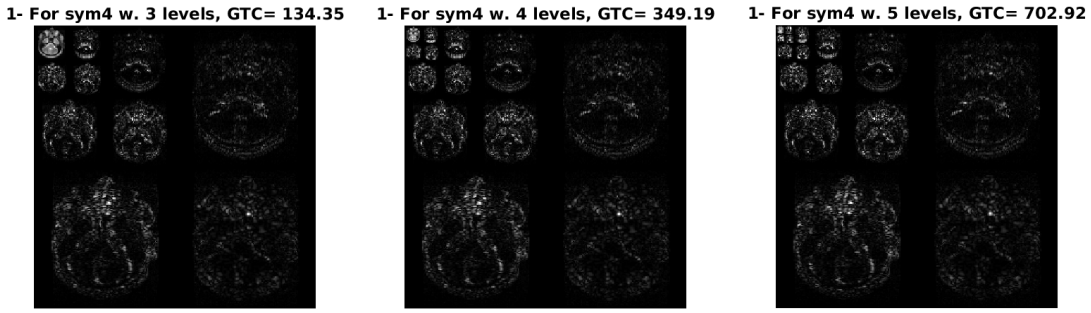

Fig. 2. Decomposição usando sym4 para a imagem 1

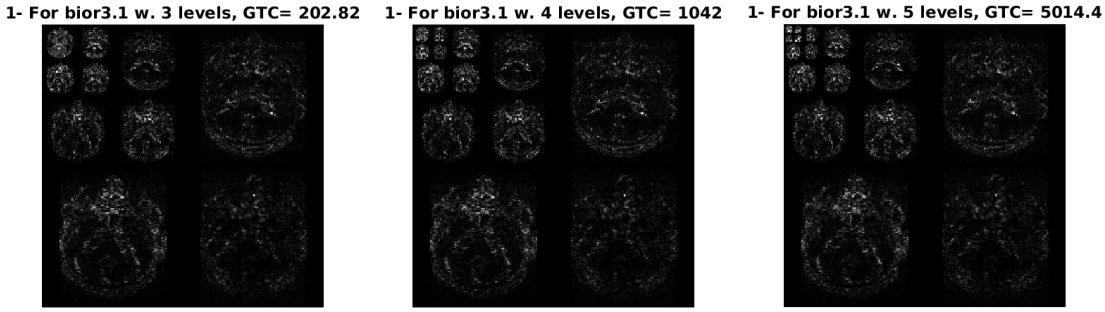

Fig. 3. Decomposição usando bior3.1 para a imagem 1

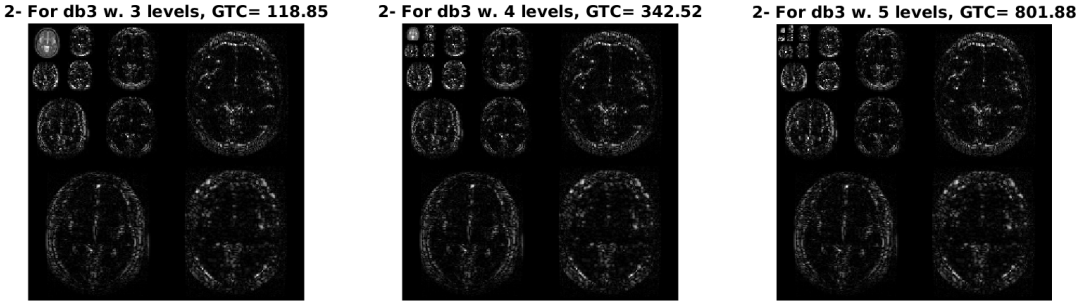

Fig. 4. Decomposição usando db3 para a imagem 2

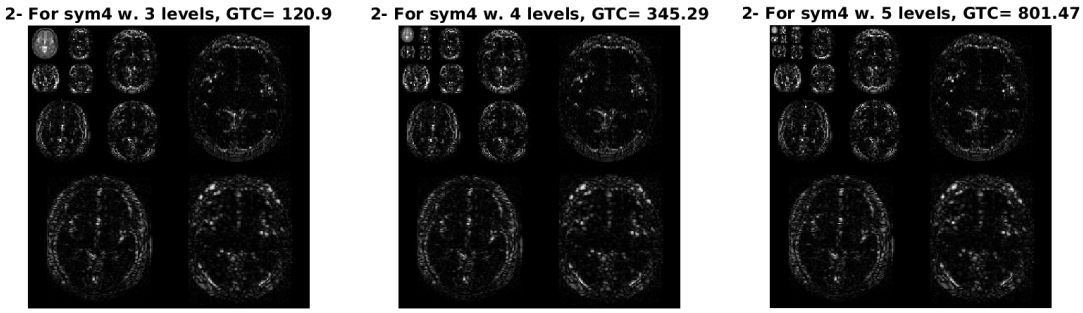

Fig. 5. Decomposição usando sym4 para a imagem 2

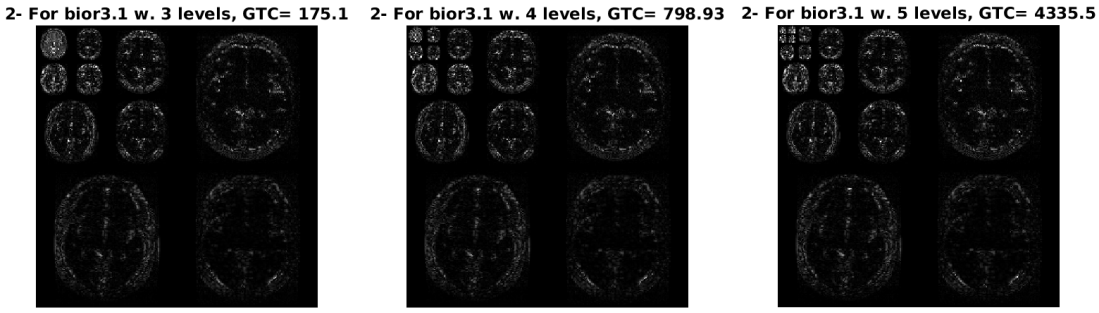

Fig. 6. Decomposição usando bior3.1 para a imagem 2

B - Compressão de Imagens: SNR vs. NM%

Baseado no G_{TC} obtido para as decomposições em diferentes níveis foi considerado que a escolha de 5 níveis seria a melhor. Para a implementação computacional foi utilizada a seguinte rotina implementada em MATLAB:

```
%% dwt2dQuantizer: Quantization for NM% coeffs in 2D-DWT
function [xq] = dwt2dQuantizer(xdc, NM)
        xq = cell(1, length(NM));
        % Obtain decomposition sizes
        sizes = cellfun(@size, xdc, 'UniformOutput', false);
        % Stack data into single array
        xdc = cellfun(@(x) x(:), xdc, 'UniformOutput', false);
        xdc = vertcat(xdc{:});

        % Sort
        [xdc_sort, I] = sort(xdc, 'descend', 'ComparisonMethod', 'abs');
        mpI = 1:1:length(xdc);
        mpI = mpI(I); % map indexes according to sorted data
        NM = round(length(xdc) * (NM/100));
        for n = 1:length(NM)
                % Quantize using Nm[%]
                tmp = zeros(length(xdc),1);
                tmp(mpI(1:NM(n))) = xdc_sort(1:NM(n));

```

```
                % Undo data stacking
                tmp_cell = cell(1,length(sizes));
                for k = 1:length(sizes)
                        nelements = prod(sizes{k});
                        tmp_cell{k} = reshape(tmp(1:nelements), sizes{k});
                        tmp(1:nelements) = [];
                end
                xq{n} = tmp_cell;
        end
end
```

Já para o procedimento de reconstrução foi implementado outra rotina para o procedimento reverso que encapsula a função de reconstrução do QMF-2D.

```
%% dwt2dReconstruction: Provides a simple wrapper to reconstruct multiple images
   using qmf_reconstruction_2
function [xr] = dwt2dReconstruction(xdc, waveletf, img_size)
        [h0, h1, g0, g1] = wfilters(waveletf);
        xr = zeros(img_size(1), img_size(2), length(xdc));
        for n = 1:length(xdc)
                [xr(:,:,n), ~, ~, ~] = qmf_reconstruction_2d(xdc{n}, h0, h1, g0, g1,
                    img_size(1), img_size(2));
        end
end
```

C - Comparativo de Resultados

As imagens 1 e 1 apresentam os resultados para as curvas de Taxa vs. Distorção. Nota-se que com a utilização das diferentes wavelets levou a diferentes resultados de performance. Em especial, para a biortogonal houve um ponto de mudança brusca na qual a adição de coeficientes levou ao melhor desempenho para todas as curvas. Em distorções maiores, a sym4 obteve melhor desempenho porém todas as curvas estão bastante próximas para os pontos considerados.

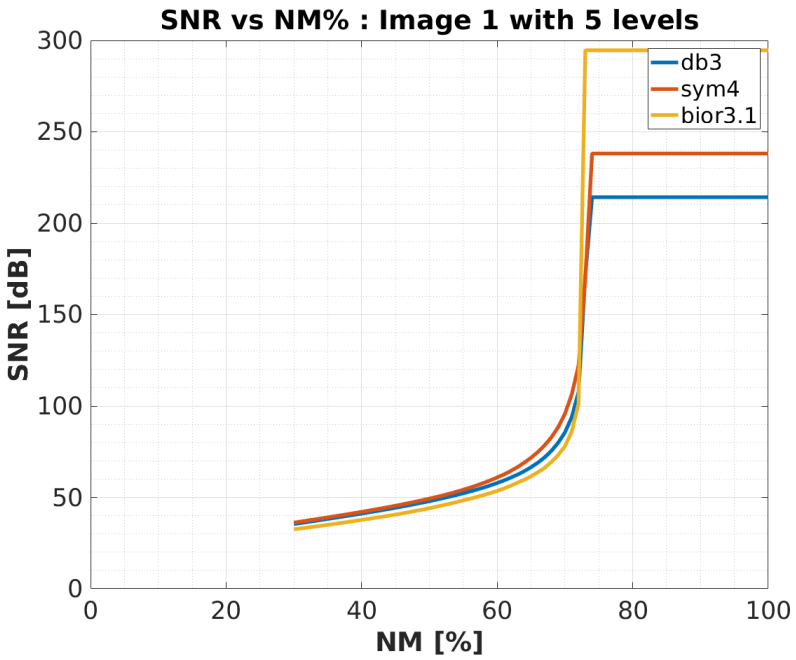

Fig. 7. Curva SNR vs. NM% para a imagem 1

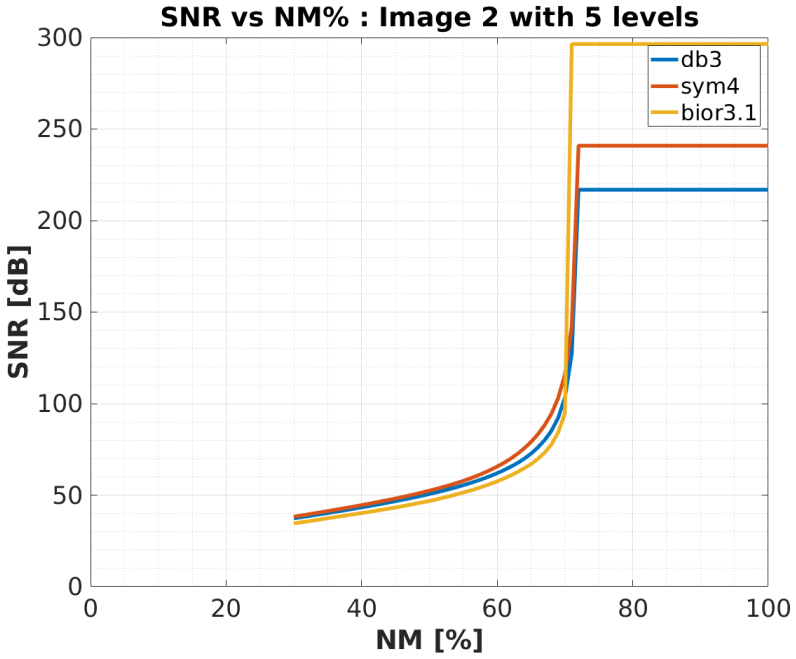

Fig. 8. Curva SNR vs. NM% para a imagem 2

Questão 2 - Detecção & Classificação de Complexos QRS em sinais de ECG

A - Implementação para detecção QRS

Baseado na implementação do algoritmo Pan-Tompkins [2], formulei uma série de simplificações e escolhas que guiaram o desenvolvimento do algoritmo. Inicialmente, realiza-se uma filtragem passa-banda entre 0.5 e 18 Hz com filtro de butterworth (*zero-phase*) para então aplicar o filtro derivativo usado no célebre trabalho original de Pan-Tompkins com 2 amostras de atraso. Posteriormente, a saída do filtro derivativo é elevada ao quadrado para ressaltar conteúdos de alta frequência presentes e é aplicado então uma média-móvel de 85 ms de duração. Com o resultado são detectados picos que estejam com distância miníma de 200 ms entre si já que está é uma condição fisiológica para o sinal considerado e são filtrados picos com valores menores que um limiar relativo a média do sinal integrado. Dessa maneira, é possível localizar bem os pontos R presentes no sinal filtrado sem que ondas acentuadas do tipo T sejam, por vezes, detectadas. Finalmente, realiza-se um refinamento dos picos R usando como referência o sinal original para que as marcações sejam efetivas. Ademais, para os pontos Q e S foram considerados os mínimos locais adjacente ao pico R obtido. Embora simples, a rotina de detecção dos pontos Q e S obteve bons resultados conforme as figuras a seguir.

Em todos os casos exceto o apresentado na figura 10 há resultados condizentes e adequatos para as anotações mostradas. No caso em que há erros o sinal apresenta inversões do pico R seguidas de picos agudos os quais foram errôneamente classificados como picos R. Para situação enunciada a marcação dos pontos Q e S também foi errônea.

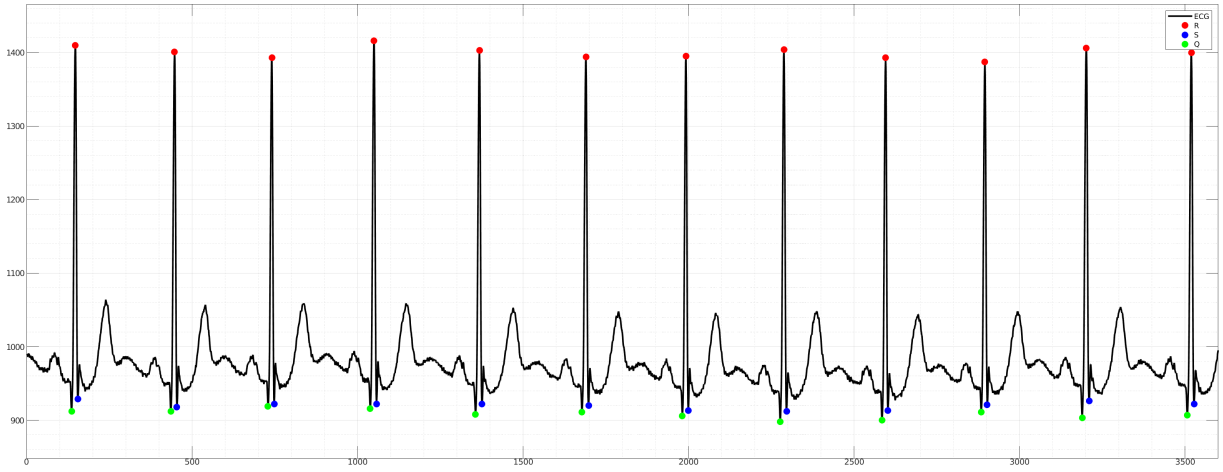

Fig. 9. Anotações obtidas para o sinal 103m6

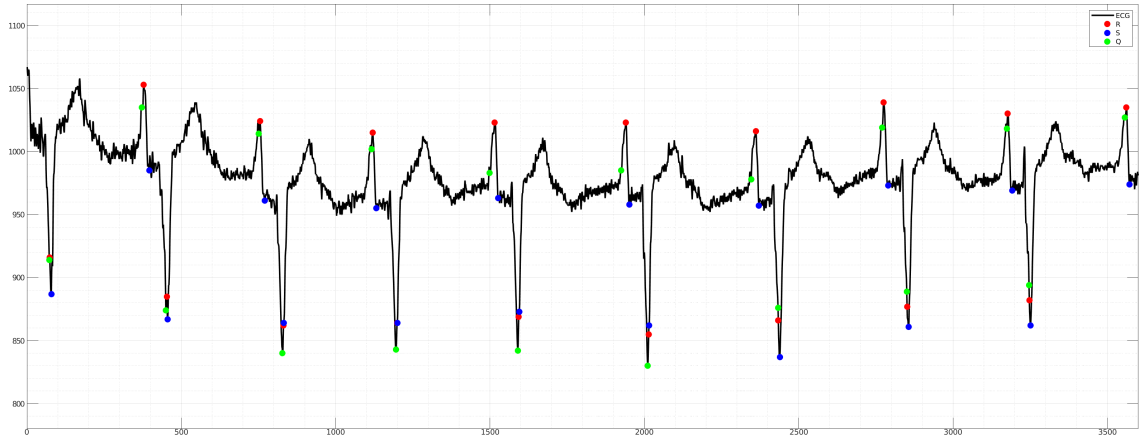

Fig. 10. Anotações obtidas para o sinal 108m3

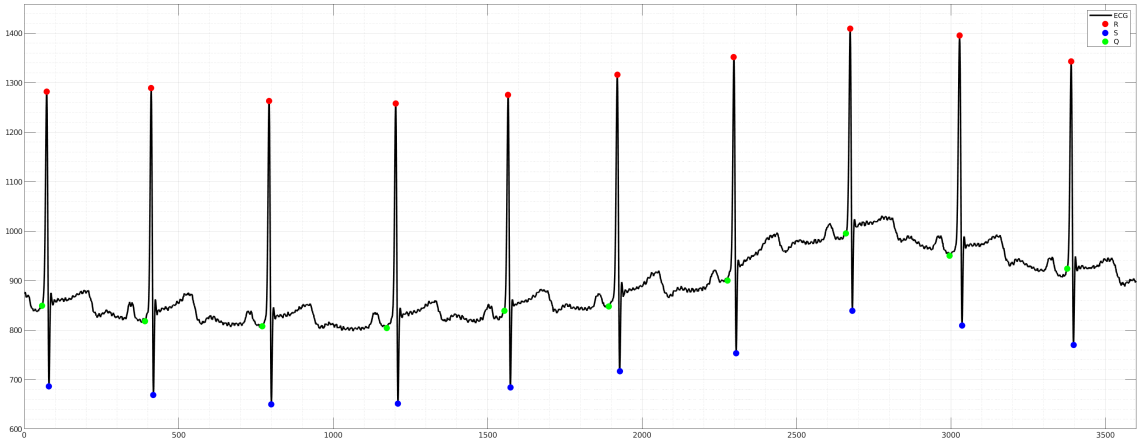

Fig. 11. Anotações obtidas para o sinal 115m6

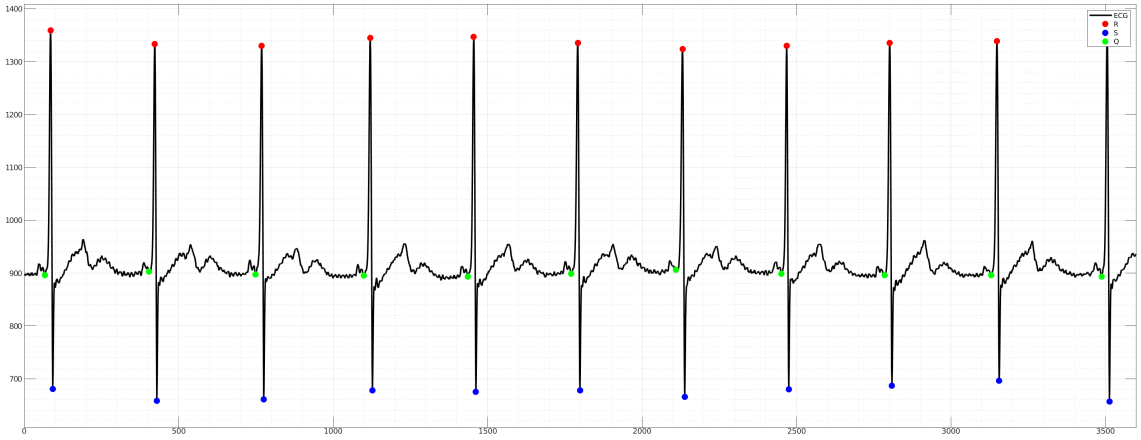

Fig. 12. Anotações obtidas para o sinal 220m3

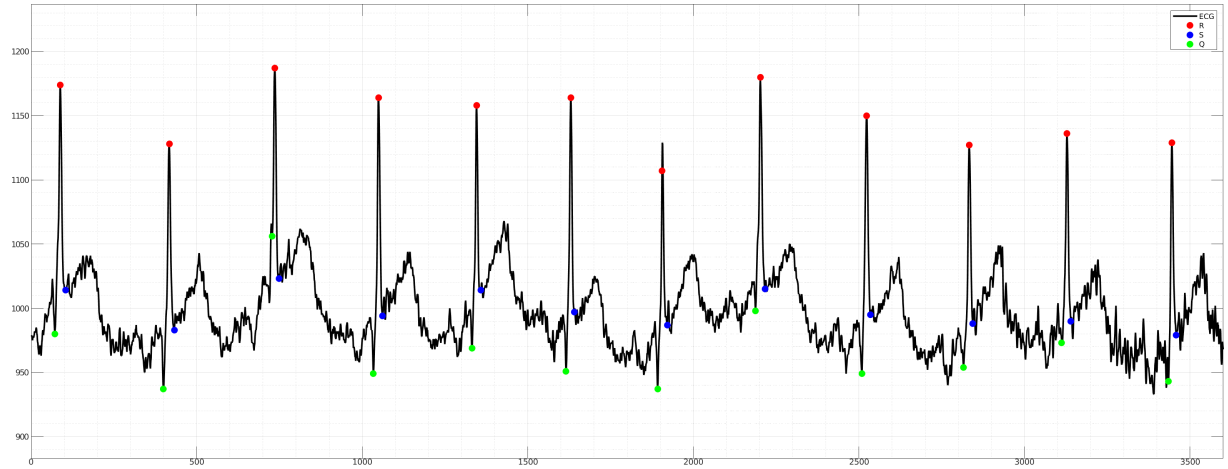

Fig. 13. Anotações obtidas para o sinal 228m1

A implementação segue a seguinte rotina:

```
%% rwave_detect: Perform R-wave detection in ECG Signals (filtering, derivative
    filtering and squaring)
function [qrs_i_raw, qrs_amp_raw] = rwave_detect(ecg, fs)
        qrs_amp_raw = [];
        qrs_i_raw = [];
        delay = 0;

        % Bandpass Filtering
        Wn = [0.5 18]*2/fs; % Cut off (based on fs)
        N = 3; % Filter Order
        [a, b] = butter(N, Wn); % Design Butterworth Filter
        ecg_f = filtfilt(a, b, ecg); % Perform Zero-Phase Filtering
        ecg_f = ecg_f / max(abs(ecg_f));

        % Derivative Filtering
        h_d = [-1 -2 0 2 1]*(1/8);
        ecg_d = conv(ecg_f, h_d); % Perform Derivative Filtering
        ecg_d = ecg_d / max(ecg_d); % Normalize Values
        delay = delay + 2; % Delay of filtering is 2 samples.

        % Squaring
        ecg_s = ecg_d.^2;

        %% Moving average
        tfrac = 0.085;
        m_d = ones(1 ,round(tfrac*fs))/round(tfrac*fs);
        ecg_m = conv(ecg_s , m_d);
        delay = delay + round((length(m_d)-1)/2);

        % Find Peaks
        % MinPeakDistance: no RR wave can occour in 200 msec time distance
        % MinPeakHeight: Threshold values based on integrated signal mean
        min_peak_h = 1.5*mean(ecg_m);
        [~,locs] = findpeaks(ecg_m, 'MINPEAKDISTANCE', round(0.2*fs), 'MinPeakHeight
            ', min_peak_h);

        locs = locs - delay;
        w = 6;
        for pt = 1:length(locs)
                ind = locs(pt)-w/2:locs(pt)+w/2;
                ecg_cut = ecg(ind);
                is_inv = mean(ecg_cut) < 0;
                if is_inv; ecg_cut = -ecg_cut; end
                R = max(ecg_cut);
                R_i = find(ecg_cut == R,1);
                locs(pt) = ind(1)-1+R_i;
        end

        qrs_amp_raw = ecg(locs);
        qrs_i_raw = locs;

        % figure('units', 'normalized', 'outerposition', [0 0 1 1]);
        % plot(ecg_f);
```

```
        % hold on;
        % plot(circshift(ecg_m,-delay));
        % stem(qrs_i_raw, ecg_f(qrs_i_raw), 'r', 'filled', 'LineStyle', 'none');
        % legend('ECG', 'M', 'R');
end
```

Para os pontos Q e S:

```
%% qswave_detect: Detect Q and S wave segments based on R location
function [q_i, s_i] = qswave_detect(ecg, r_i) %TODO: optimize and refactor try catch
        s_i = zeros(length(r_i), 1);
        q_i = zeros(length(r_i), 1);
        th = 1;
        %% QS Wave Detect
        for pt = 1:length(r_i)
                % S
                n = 1;
                try
                        while (ecg(r_i(pt)+n) + th) >= ecg(r_i(pt)+n+1)
                                n = n + 1;
                        end
                        s_i(pt) = r_i(pt)+n;
                catch ME
                        switch ME.identifier
                                case 'MATLAB:badsubscript' % R points near the end
                                    of the signal could trigger bad subscript (maybe?
                                     idk)
                                        continue;
                                otherwise
                                        rethrow(ME);
                        end
                end
                % R
                try
                        n = -1;
                        while (ecg(r_i(pt)+n) + th) >= ecg(r_i(pt)+n-1)
                                n = n - 1;
                        end
                        q_i(pt) = r_i(pt)+n;
                catch ME
                        switch ME.identifier
                                case 'MATLAB:badsubscript'
                                        continue;
                                otherwise
                                        rethrow(ME);
                        end
                end
        end
end
```

C - Classificação de QRS com SVM

As tentativas iniciais revelaram que a utilização de *kernel* polinomial ou RBF (*Radial Basis Function*) para as condições de teste geravam excessivo *overfitting* no classificador. Com a utilização de um *kernel* linear foram obtidos resultados interessante com a metodologia de treinamento e validação utilizada.

Para a seleção do conjunto de validação são selecionados 30% dos sinais de maneira aleatória para extração dos pontos de interesse. O restante dos sinais é utilizado para treinamento de maneira que entre os conjuntos de validação e treinamento não existam dados em comum. Para aumentar a ocorrência de casos positivos no conjunto de treinamento foi utilizado uma sobreposição de 50% entre as janelas. Como tolerância para o centro da janela foi escolhida uma distância de 40 amostras (11.1 ms). A entrada da SVM recebe janelas de 1 segundo de duração. Ademais, não houve sobreposição de janelas nas condições de validação, algo que gerou poucos casos positivos no conjunto de validação.

Novamente, para reduzir a possibilidade de *overfitting* no modelo foi atribuido um Custo de 8 para falsos negativos e 1 para falsos positivos já que existem bem menos casos positivos que negativos é desejado que, em detrimento de falsos positivos, o classificador não obtenha falsos negativos durante o processo de detecção.

Foram testados cada um dos picos Q, R e S em sessão única de treinamento e validação e os resultados obtidos são mostrados nas figuras a seguir. É válido ressaltar que para todas as métricas mostradas de *FP-Rate* o valor elevado é decorrente do pequeno conjunto de casos positivos já que não foi realizado overlap na validação. Isso é notório pelo fato de que TP-Rate ainda se mantém elevado.

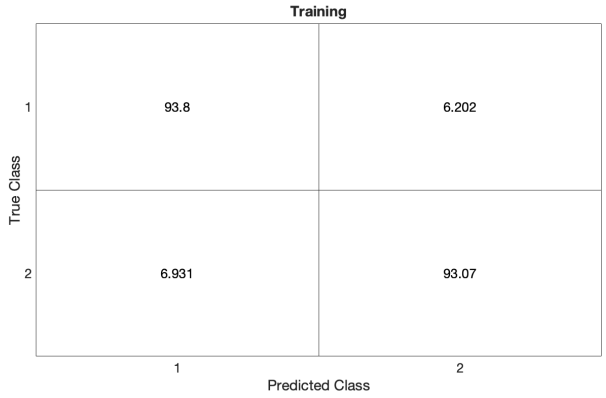

Fig. 14. Matriz de Confusão para Treinamento da Classificação do Pico R (valores percentuais)

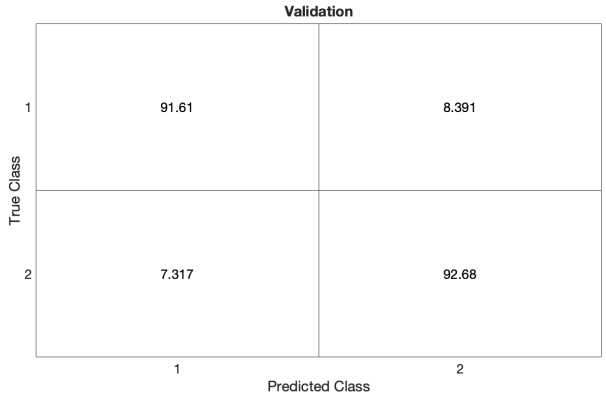

Fig. 15. Matriz de Confusão para Validação da Classificação do Pico R (valores percentuais)

```
Validation Results:
FP Rate is 34.86
TP Rate is 98.67
Precision is 91.61
Accuracy is 91.76
Recall is 98.67
F-Measure is 95.01
```

Fig. 16. Métricas para a sessão de Validação do Pico R

```
Validation Results:
FP Rate is 50.82
TP Rate is 96.7
Precision is 87.4
Accuracy is 86.47
Recall is 96.7
F-Measure is 91.81
```

Fig. 17. Métricas para a sessão de Validação do Pico Q

```
Validation Results:
FP Rate is 47.09
TP Rate is 97.73
Precision is 87.89
Accuracy is 87.76
Recall is 97.73
F-Measure is 92.55
```

Fig. 18. Métricas para a sessão de Validação do Pico S

D - Detecção de QRS com SVM

Para a etapa final da detecção utilizei a classificação dos pontos R para a qual foram obtidos melhores resultados. Realizei a marção em algumas janelas e obtive os valores R selecionando pontos de máximo. Como em algum sinais há inversão do pico R, houveram marcações errôneas para o procedimento citado. Abaixo, são exibidas as marcações obtidas (vide figura 19).

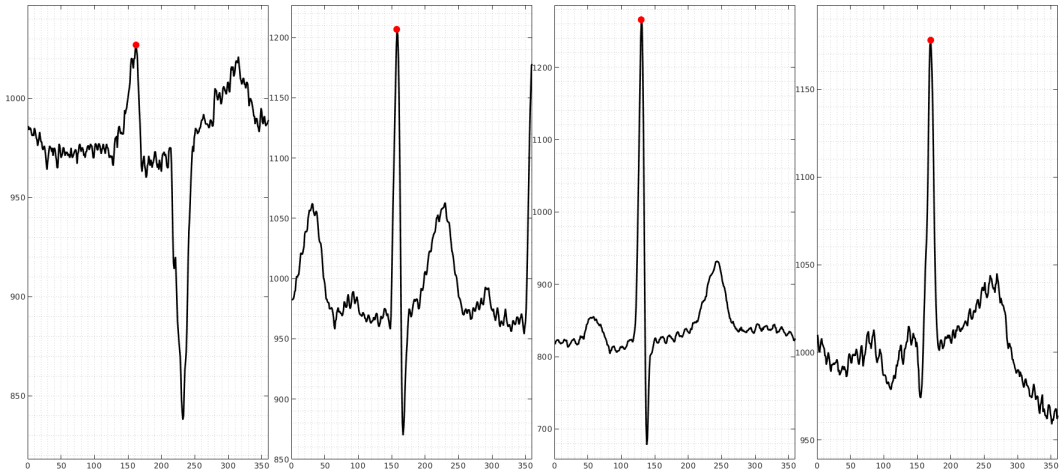

Fig. 19. Marcações para o ponto R em diferentes janelas.

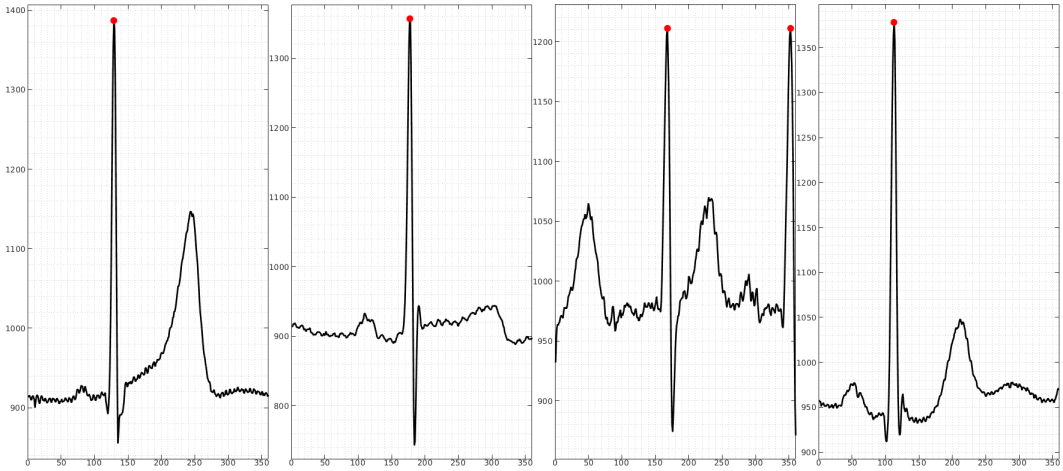

Fig. 20. Marcações para o ponto R em diferentes janelas.

Considerações Finais

Considero que o escopo da matéria foi bastante extenso e me possibilitou sistematizar conteúdos e explorar implementações que não havia feito antes e que foram bastante divertidas em sua grande maioria! Além disso, o conteúdo foi cumprido de maneira excepcional já que foi possível esclarecer todos os tópicos que tive dificuldade com o auxílio das aulas presenciais ou por meio dos trabalhos práticos desenvolvidos na prova 1 e 2. Finalmente, agradeço por toda a ajuda prestada ao longo do semestre e pelas aulas.

Muito Obrigado, **Davi Mendes**.

2. REFERENCES

[1] N.S. Jayant, P. Noll, Digital Coding of Waveforms, Prentice-Hall, 1984.

[2] Pan, J., Tompkins, W.J.: A real-time QRS detection algorithm. IEEE Trans. Biomed. Eng. 3, 230–236 (1985)

Anexos

Questão 1 - Implementação completa

```
clear all; close all; clc;

% Add Filterbanks2D to PATH
addpath('../filterbanks2d');
rehash path;

% P2 Signals Path
joinDataPath = @(file) ['/home/davi/Dropbox/processamento_sinais_biologicos_02_2019/
    sinais_prova_2/' file];

% Load Images
load(joinDataPath('example2.mat'), 'x');
levels = 3:1:5;
imgs = {x{6}, x{12}};
waveletf = {'db3', 'sym4', 'bior3.1'};
xd = cell(length(levels), length(imgs), length(waveletf));
xdc = cell(length(levels), length(imgs), length(waveletf));

% A)
try % Load saved data to improve runtime execution
        load('Q1.mat');
catch ME
        for im = 1:length(imgs)
                for wf = 1:length(waveletf)
                        fig = figure('units', 'normalized', 'outerposition', [0 0 1
                            1]);
                        for l = 1:length(levels)
                                [h0, h1, g0, g1] = wfilters(waveletf{wf});
                                [xd{l,im,wf}, xdc{l,im,wf}, ~] =
                                    qmf_decomposition_2d(imgs{im}, h0, h1, levels(l))
                                    ;
                                subplot(1,length(levels),l);
                                imshow(xd{l,im,wf}, []);
                                gtc = transformCodingGain(xdc{l,im,wf});
                                im_title = [int2str(im) '- For ' waveletf{wf} ' w. '
                                     int2str(levels(l)) ' levels, GTC= ' num2str(gtc
                                    ,5)];
                                title(im_title);
                                disp(im_title);
                        end
                        truesize(fig);
                        % saveas(fig, [int2str(im) '_' waveletf{wf} '.png']);
                end
        end
        save('Q1.mat', 'xd', 'xdc');
end

% B)
NM = 30:100;
chosen_level = 5;
```

```
ilvl = find(levels == chosen_level);
for im = 1:length(imgs)
        figure('units', 'normalized', 'outerposition', [0 0 1 1]);
        for wf = 1:length(levels)
                im_title = ['SNR vs NM% : Image ' int2str(im) ' with ' int2str(
                    chosen_level) ' levels'];
                [xq] = dwt2dQuantizer(xdc{ilvl,im,wf}, NM);
                xr = dwt2dReconstruction(xq, waveletf{wf}, size(imgs{im}));
                SNR = evaluateSNR(xr, imgs{im});
                plot(NM, SNR, 'LineWidth', 4);
                hold on;
        end
        title(im_title);
        legend(waveletf);
        xlim([0 100]);
        grid on;
        grid minor;
        set(get(gca,'YLabel'),'String','SNR [dB]');
        set(get(gca,'XLabel'),'String','NM [%]');
        set(findall(gcf,'type','text'), 'FontSize', 32, 'fontWeight', 'bold');
        set(gca,'FontSize', 26);
end

%% evaluateSNR: function description
function [snr_vals] = evaluateSNR(rimg, img)
        snr_vals = zeros(1, size(rimg,3));
        SNR = @(n_data, ref_data) mean( n_data(:).^ 2 ) / mean( (n_data(:) -
            ref_data(:)).^2 );
        for s = 1:size(rimg,3)
                snr_vals(s) = SNR(rimg(:,:,s), img);
        end
        snr_vals = 10*log10(snr_vals);
end
```

Questão 2 - Implementação completa

```
clear all; close all; clc;

% Add Functions to PATH
addpath('../EXER1-ECG_DEP');
addpath('../utils_P1');
addpath('../P1');
rehash path;

% ECG Signals
ecg_signals_path = '~/Dropbox/processamento_sinais_biologicos_02_2019/sinais_prova_2
    /ecg/';
signalfield_name = 'val';

% Load Signals
[ecg_signals, fs, s_name] = loadmat(ecg_signals_path, signalfield_name);
fs = 360;
rand_signal = randi(length(ecg_signals));
```

```
% rand_signal = 14;
disp(['Using Signal: ' s_name{rand_signal} ' Number: ' int2str(rand_signal)]);
ecg = ecg_signals{rand_signal}(:);

% Perform R-Wave Detection
[qrs_i, qrs_amp] = rwave_detect(ecg, fs);
[pan_qrs_amp, pan_qrs_i, ~] = pan_tompkin(ecg,fs, 0);

disp(['Reference QRS Indexes contains ' int2str(length(pan_qrs_i)) ' points.']);
disp(['Obtained QRS Indexes contains ' int2str(length(qrs_i)) ' points.']);
disp(['--- ' int2str(length(find(qrs_i == pan_qrs_i))) ' points in both sets ---']);
disp('Me:');
disp(qrs_i);
disp('Pan-Tompkins:');
disp(pan_qrs_i(:));

% Use QS wave detect
[q_i, s_i] = qswave_detect(ecg, qrs_i);

figure('units', 'normalized', 'outerposition', [0 0 1 1]);
plot(ecg,'k', 'LineWidth', 2);
ylim([min(ecg)-50  max(ecg)+50]);
xlim([0 length(ecg)]);
grid on;
grid minor;
hold on;
stem(qrs_i, qrs_amp, 'r', 'filled', 'LineStyle', 'none', 'MarkerSize', 8);
stem(s_i, ecg(s_i), 'b', 'filled', 'LineStyle', 'none', 'MarkerSize', 8);
stem(q_i, ecg(q_i), 'g', 'filled', 'LineStyle', 'none', 'MarkerSize', 8);
legend('ECG', 'R', 'S', 'Q');
```

Questão 2 - Implementação com SVM

```
clear all; close all; clc;

% Add Functions to PATH
addpath('../EXER1-ECG_DEP');
addpath('../utils_P1');
addpath('../P1');
rehash path;

% ECG Signals
ecg_signals_path = '~/Dropbox/processamento_sinais_biologicos_02_2019/sinais_prova_2
    /ecg/';
signalfield_name = 'val';

% Load Signals
[ecg_signals, fs, s_name] = loadmat(ecg_signals_path, signalfield_name);
if isempty(fs); fs = 360; end
s_len = length(ecg_signals);

% Extract QRS Indexes
q_i = cell(1, s_len);
```

```
r_i = cell(1, s_len);
s_i = cell(1, s_len);
for n = 1:s_len
        [r_i{n}, ~] = rwave_detect(ecg_signals{n}, fs);
        [q_i{n}, s_i{n}] = qswave_detect(ecg_signals{n}, r_i{n});
end

% SVM Params
alp = 0.7; % Percentage of data to the training stage
tol_win = 40; % Tolerance for the QRS position in window
win_samples = 1*fs;
train_overlap = 80; % Percentage of overlap in training windows
train_i = r_i;

% Extract Features
rand_set = randperm(s_len);
val_set_index = rand_set(1:round((1-alp)*s_len));
train_set_index = setdiff(rand_set,val_set_index);
% Training Features
[train_features, train_class] = getTrainFeatures(ecg_signals, train_set_index,
    train_i, win_samples, train_overlap, tol_win);
% Validation Features (same as train features but with no overlap)
[val_features, val_class] = getTrainFeatures(ecg_signals, val_set_index, train_i,
    win_samples, 80, tol_win);

% Perform Classification
svm = fitcsvm(train_features, train_class, 'KernelFunction', 'linear', 'Standardize'
    , true, 'ClassNames', [0,1], 'Cost', [0, 1; 8, 0]);
% out_train = predict(svm, train_features);
out_val = predict(svm, val_features);

% Plot Confusion Matrix
% confusionChart(confusionmat(train_class, out_train), 'Training');
confusionChart(confusionmat(val_class, out_val), 'Validation');

% Get Session Results
disp('Validation Results:');
sessionResults(confusionmat(val_class, out_val));



%% sessionResults: Obtain ROC Metrics
function sessionResults(confusion_matrix)
        % Metrics
        fpRate = @(cm) cm(1,2)/sum(cm(:,2));
        tpRate = @(cm) cm(1,1)/sum(cm(:,1));
        precision = @(cm) cm(1,1)/sum(cm(1,:));
        accuracy = @(cm) sum(diag(cm))/sum(cm(:));
        recall = @(cm) cm(1,1)/sum(cm(:,1));
        fMeasure = @(cm) 2/( 1/precision(cm) + 1/recall(cm) );

        % Display in console
        disp(['FP Rate is ' num2str(100*fpRate(confusion_matrix),4)]);
        disp(['TP Rate is ' num2str(100*tpRate(confusion_matrix),4)]);
```

```
        disp(['Precision is ' num2str(100*precision(confusion_matrix),4)]);
        disp(['Accuracy is ' num2str(100*accuracy(confusion_matrix),4)]);
        disp(['Recall is ' num2str(100*recall(confusion_matrix),4)]);
        disp(['F-Measure is ' num2str(100*fMeasure(confusion_matrix),4)]);
        disp(' ');
end

%% confusionChart: Plot Confusion Matrix Chart
function [varargout] = confusionChart(C, chart_title)
        C(1,:) = 100*(C(1,:)/sum(C(1,:)));
        C(2,:) = 100*(C(2,:)/sum(C(2,:)));
        figure('units','normalized','outerposition',[0 0 1 1]);
        h = heatmap(C);
        h.ColorbarVisible = 'off';
        colormap white;
        title(chart_title);
        h.YLabel = 'True Class';
        h.XLabel = 'Predicted Class';
        set(findall(gcf,'type','text'), 'FontSize', 32, 'fontWeight', 'bold');
        set(gca,'FontSize',26);
        if nargout == 1
                varargout{1} = h;
        end
end
```