

# Trabalho Prático - Processamento Digital de Sinal



*Corte de ruído presente num sinal de fala.*

*Relatório do Trabalho Prático de Processamento Digital de Sinal.*



Pedro Diogo. 52586

*20 de Julho de 2013*

# Índice

<b>Introdução</b>	<b>3</b>
<b>Concepção</b>	<b>4</b>
<i>Algoritmo</i>	<i>4</i>
<i>Análise Experimental</i>	<i>4</i>
<i>Implementação</i>	<i>8</i>
<b>Resultados Obtidos e Análise Crítica</b>	<b>12</b>
<i>Conclusão</i>	<i>12</i>

# Introdução

O trabalho proposto consiste na captação de um sinal audio (fala), pelo microfone do computador usando *MatLab*, para posterior corte da componente de ruído. A componente de corte de ruído baseia-se em estatísticas e o algoritmo usado tem como base um *threshold* (valor limite) típico para um dado SNR (relação sinal-ruído): valores acima devem ser considerados informação, enquanto que aqueles abaixo podem ser descartados, isto é, considerados ruído.

Neste relatório será então apresentada a solução ao problema, detalhando todos os passos tomados no seu desenvolvimento (análise prévia experimental e implementação de algoritmo) realçando o porquê dessas decisões e apresentando o código (script e funções) *MatLab* concebidas para o problema.

# Concepção

## Algoritmo

O algoritmo para corte de ruído usado foi baseado em estatísticas do ruído (variância e média), definindo assim um *threshold* que decide se o sinal lido contém ou não ruído. Para isto, é preciso conhecer o SNR da mensagem a analisar para aplicar o *threshold* correto. Este *threshold* correto provém da análise prévia experimental a diferentes mensagens audio, tendo como parâmetro um valor *alpha*.

```
threshold = media_ruído + alpha * variancia_ruído;
```

De seguida é apresentada a forma como foi obtido o valor de *alpha* correto para diferentes valores de SNR.

## Análise Experimental

Para se saber qual o melhor valor *alpha* a usar na mensagem, foi feito um estudo a várias mensagens com diferentes ruídos. O método usado foi bastante simples: gravar uma mensagem audio “original.wav” e somar valores *random* a esse ficheiro audio, com diferentes amplitudes (*fator\_ruído*), para obter diferentes valores SNR.

```
%Induzir erro no sinal original  
fator_ruído = 0.0001 %alterado manualmente para se obter diferentes SNR  
sinal_alterado = audio+(randn(length(audio),1).*fator_ruído);
```

O SNR calculado foi aquele correspondente a apenas ao primeiro segundo da mensagem da fala - desta forma, consegue-se analisar apenas o ruído de fundo induzido na mensagem, sem qualquer fala. As variáveis *media\_ruído* e *variancia\_ruído* acima representadas correspondem, portanto, ao primeiro segundo da mensagem audio e serão usadas para o cálculo do SNR.

No final desta fase experimental, guardou-se cada um desses ficheiros .wav respetivos de cada SNR, ficando, no final, com um total de 9 valores diferentes de SNR.

A função *Matlab* responsável por induzir este erro foi chamada de *snr\_pds*:

```

function [ SNR ] = snr_pds( audio )

%Induzir erro no sinal original
fator_ruído = 0.0001 %alterado manualmente para se obter diferentes SNR
sinal_alterado = audio+(randn(length(audio),1).*fator_ruído);

disp('Vai tocar sinal com ruído')
sound(sinal_alterado, 44100, 16);
figure(2);
title('Sinal Alterado');
plot(sinal_alterado);

% Gravação em disco
%wavwrite(sinal_alterado, 44100, 'snr25');

j=1;
sinal=0;

%seleção da parte de sinal (1 segundo)
ruído = sinal_alterado(1:round(1/(1/44100)));

r=((std(ruído))^2)+((mean(ruído))^2);%potência de ruído

sumatorio=0;
for(i=1:length(audio))
    sumatorio=sumatorio+audio(i).^2;
end
s=sqrt((1/(length(audio)))*sumatorio); %media quadratica do sinal

SNR=10*log10(((s-r)/r))

end %Fim da função snr_pds

```

Tendo os vários ficheiros com diferentes valores de SNR, passou-se à análise de qual o melhor *alpha* (parâmetro do *threshold* de ruído) a usar para remover o ruído adicionado antes. Após várias experiências, conclui-se que, para os dados valores de SNR, deve-se usar o *alpha* correspondente da tabela:

SNR	alpha
8 - 10	1.8
10.1 - 12	1.9
12.1 - 14	2
14.1 - 16	2.2
16.1 - 18	2.3
18.1 - 20	2.4

Para descobrir qual o melhor *alpha* a ser usado para remover o ruído, foi usado um algoritmo baseado em janelas, em que se compara cada amostra do audio com o valor de *threshold* e apenas se considera fala aquela janela que contém, pelo menos, 10% dos valores acima do *threshold*. Esta forma é mais fiável do que apenas comparar cada amostra do conteúdo audio. O tamanho de mensagem escolhido foi 30. A função *alpha\_pds* usada para determinar a tabela acima apresenta-se de seguida:

```

function [] = alpha_pds( audio )

alpha = 1.8 %alpha alterado constantemente para determinar, experimentalmente,
qual o melhor para um dado SNR.

ruído = audio(1:round(0.6/(1/44100)));
figure(2);
plot(ruído);
title('RUIDO de 0.6 segundos do original');
xlabel('n');
ylabel('AMPLITUDE');

media_ruído = mean(ruído); %media
variancia_ruído = std(ruído); %desvio padrao
threshold = media_ruído + alpha * variancia_ruído; %threshold para
%determinar se info ou ruído
threshold_abaxo=0;
threshold_acima=0;
tam_janela = 30;
fala = [];
ruídoapa = [];
j=1;

for i=1:30:length(audio)
    if length(audio)-i<29
        aux=length(audio)-i;
    else aux = 29;
    end
    for j=i:1:i+aux
        if(abs(audio(j)) > abs(threshold))
            threshold_acima = threshold_acima + 1;
        else
            threshold_abaxo = threshold_abaxo + 1;
        end
    end
    if(threshold_acima >= 0.10*tam_janela) %guardo a fala
        fala = [fala; audio(i:i+aux)];
    else
        ruídoapa = [fala; audio(i:i+aux)]; %caso queira verificar
    end
    threshold_acima = 0;
    threshold_abaxo = 0;
end

pause(2);
figure(4);
plot(fala);
title('INFORMACAO?');
xlabel('n');
ylabel('AMPLITUDE');
pause(5);
disp('A ouvir sinal recuperado...')
sound(fala, 44100, 16);

end %Fim função alpha_pds

```

## Implementação

Tendo a tabela indicando qual o valor *alpha* a ser usado na fórmula do *threshold* para um dado SNR, implementou-se o mesmo algoritmo apresentado na função anterior *alpha\_pds* tendo apenas em conta a tabela obtida anteriormente pela análise experimental. Desta forma, ficou *hard-coded* em *MatLab* qual o valor de *alpha* correto para um dado SNR.

Assim sendo, a função para aplicar o algoritmo chamada de *limpa\_audio*:



```

function [] = limpa_audio( audio )

% 1º - Determinar SNR (ver 0.6seg do audio para ruído);
% 2º - Determinar qual alpha devo usar;
% 3º - Aplicar algoritmo com threshold e alpha decidido em cima.

% Seleção da parte de sinal (0.6 segundo)
ruído = audio(1:round(0.6/(1/44100)));
media_ruído = mean(ruído);
variância_ruído = std(ruído);
r=((std(ruído))^2)+((mean(ruído))^2);%potência de ruído

sumatorio=0;
for(i=1:length(audio))
    sumatorio=sumatorio+audio(i).^2;
end
s=sqrt((1/(length(audio)))*sumatorio); %media quadratica do sinal

SNR=10*log10(((s-r)/r));
SNR=SNR-2.5;
alpha =0;
aux=0;

if SNR <8
    disp('impossível recuperar sinal');
else
    if SNR >=8 && SNR <10
        alpha = 1.8;
    else
        if SNR >=10 && SNR <12
            alpha = 1.9;
        else
            if SNR >=12 && SNR <14
                alpha = 2;
            else
                if SNR >=14 && SNR <16
                    alpha = 2.2;
                else
                    if SNR >=16 && SNR <18
                        alpha = 2.3;
                    else
                        if SNR >=18 && SNR <22
                            alpha = 2.45;
                        end
                    end
                end
            end
        end
    end
end
end
end
end
end
end
end

```

```

disp('SNR detetado =');disp(SNR);
disp('alpha a utilizar =');disp(alpha);

threshold = media_ruido + alpha * variancia_ruido;
threshold_acima = 0;
threshold_abaixo = 0;
tam_janela=30;
fala = [];
ruidoapa = [];

for i=1:30:length(audio)
    if length(audio)-i<29
        aux=length(audio)-i;
    else aux = 29;
    end
    for j=i:1:i+aux
        if(abs(audio(j)) > abs(threshold))
            threshold_acima = threshold_acima + 1;
        else
            threshold_abaixo = threshold_abaixo + 1;
        end
    end
    if(threshold_acima >= 0.15*tam_janela) %guardo a fala
        fala = [fala; audio(i:i+aux)];
    else
        ruidoapa = [fala; audio(i:i+aux)]; %útil para debug
    end
    threshold_acima = 0;
    threshold_abaixo = 0;
end

figure(4);
plot(fala);
title('INFORMACAO');
xlabel('n');
ylabel('AMPLITUDE');

pause(3);
disp('A ouvir sinal recuperado...')
sound(fala, 44100, 16);

end

% Fim função limpa_audio

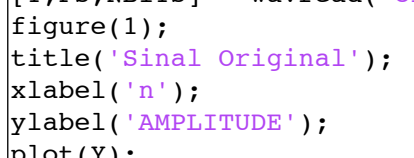
```

Por fim, a script *PDS\_TPF.m* que invoca todas estas funções para chegar à conclusão. Como os diferentes ficheiros audio com diferentes SNR já foram gerados e guardados, não há necessidade de os invocar de novo, pelo que foi comentado o código desnecessário e apenas é invocada a função *limpa\_audio*, passando-lhe como argumento de entrada o sinal audio com ruído induzido. De seguida apresenta-se então a *script*:

```
fs = 44100;
bits = 16;
channel = 1;
device = 0;
tempo = 4;

disp('Início');

% -- Zona comentada porque já gravei
% a mensagem usada para todo o processo --
%recObj = audiorecorder(fs, bits, channel,0);
%get(recObj)
%disp('Start speaking.')
%recordblocking(recObj, tempo);
%disp('End of Recording.');
```



```
%play(recObj);
%original = getaudiodata(recObj); % guardo audio num array
%wavwrite(original, 44100, 'original');

%Leitura do ficheiro original, sem ruído induzido.
[Y,FS,NBITS] = wavread('original');
figure(1);
title('Sinal Original');
xlabel('n');
ylabel('AMPLITUDE');
plot(Y);

% -- Função para induzir ruído e determinar SNR resultante --
% [SNR] = snr_pds(Y);

[Y2,FS,NBITS] = wavread('snr8_23');
pause(5)
disp('A ouvir original com um dado SNR')
sound(Y2, 44100, 16);

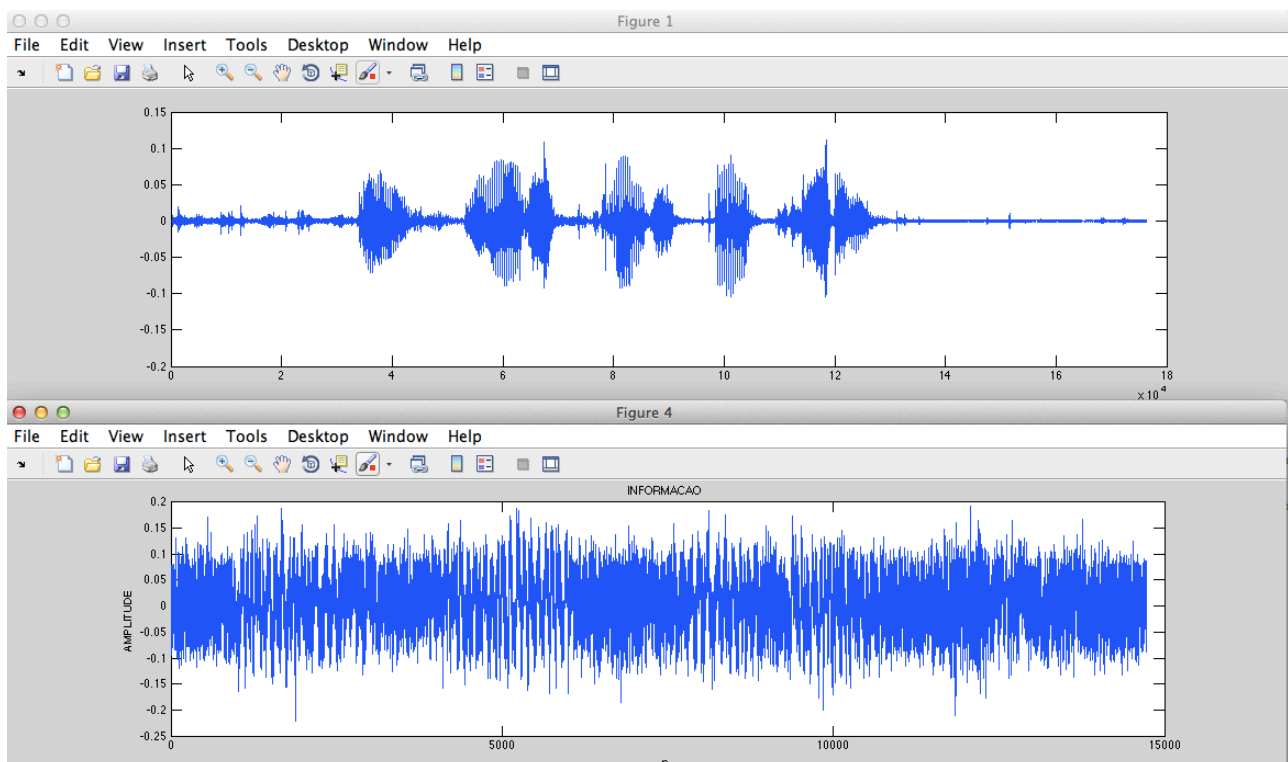
% -- Função para descobrir alpha correto, dado um SNR --
% alpha_pds(Y2);

%-- Função para descobrir alpha correto, dado um SNR --
limpa_audio(Y2);
```

## Resultados Obtidos e Análise Crítica

Apesar de o algoritmo funcionar, isto é, cortar as partes do áudio que foram consideradas ruído (os primeiros 0.6 segundos, como foi visto antes), o processo não é o mais indicado para aplicações mais exigentes. Como trabalho futuro, poderia-se melhorar o algoritmo, introduzindo por cima outros algoritmos como *Zero Crossing Rate*. Além disso, este algoritmo pressupõe que o ruído presente na mensagem é estático, isto é, irá corresponder maioritariamente à aquele presente nos primeiros 0.6 segundos da mensagem.

Como se pode ver na figura, as componentes em que não há fala são completamente cortadas, havendo uma grande compactação do áudio.



## Conclusão

Conclui-se assim que o problema proposto foi realizado com sucesso, aplicando vários estudos prévios (análise experimental) para posterior implementação do algoritmo baseado em estatísticas.