```
▶ PlotlyBackend()
```

```
begin
    using Plots
    using DSP
    using MAT
    using SampledSignals
    using Pkg
    using Statistics
    plotly()
end
```

Gabriel Tavares Ferrarez

Nusp: 10773801

Exercicio 1

```
begin
    desejado1_file = matopen("desejado1.mat");
    desejado1 = read(desejado1_file, "d1")[1,:];
    close(desejado1_file);

    entrada1_file = matopen("entrada1.mat");
    entrada1 = read(entrada1_file, "u1")[1,:];
    close(entrada1_file);
end
```

Primeiro vamos testar o algoritimo com um valor grande de coeficientes e com um valor pequeno de passo de adaptação.

M = 200 coeficientes

μ = 0.0001

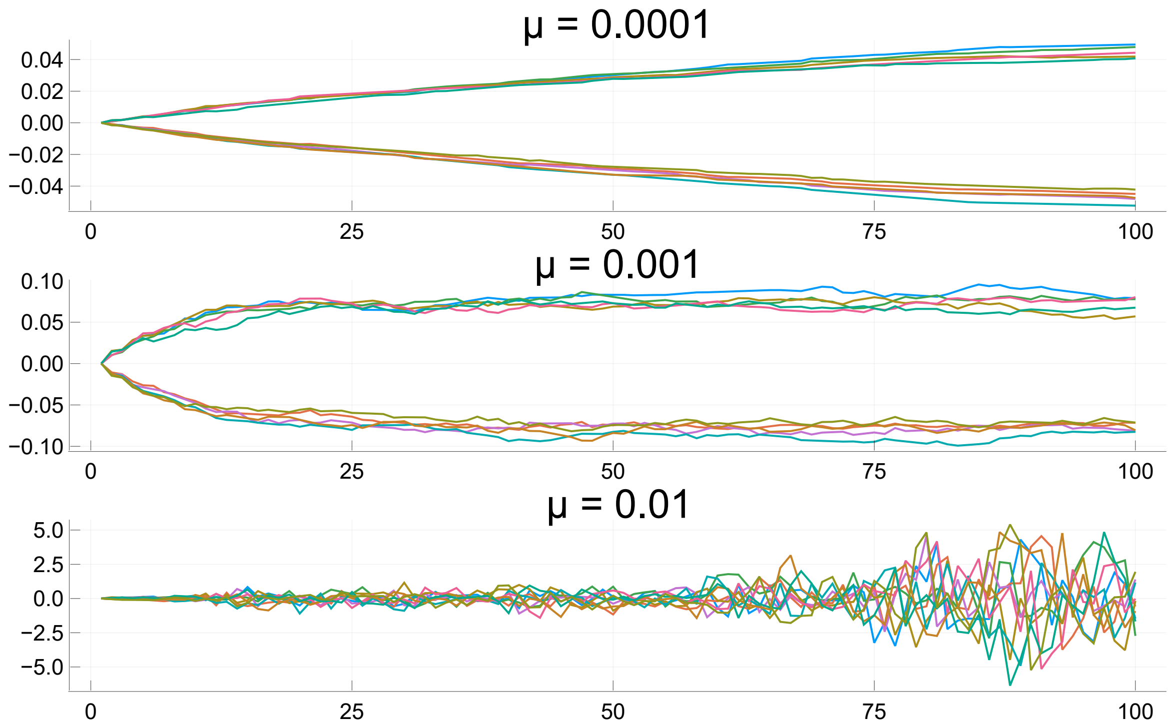
Vamos analisar se os coeficientes estão convergindo e quantos dos coeficiente são muito pequenos e vão influenciar menos na estimação do filtro.

```
begin
    M200 = 200 #num filtros
    N = length(entrada1) #num interações
    noprint
end
```

Estimação do passo de convergência

Nessa etapa vamos plotar 10 dos coeficientes e ver como eles se comportam com a mudança de μ.

```
begin
    μ_peq = 1e-4
    μ_med = 1e-3
    μ_grande = 1e-2
    W_peq, erro_peq =  LMS(entrada1, desejado1, M200,N,μ_peq)
    W_med, erro_med =  LMS(entrada1, desejado1, M200,N,μ_med)
    W_grande, erro_grande =  LMS(entrada1, desejado1, M200,N,μ_grande)
    noprint
end
```

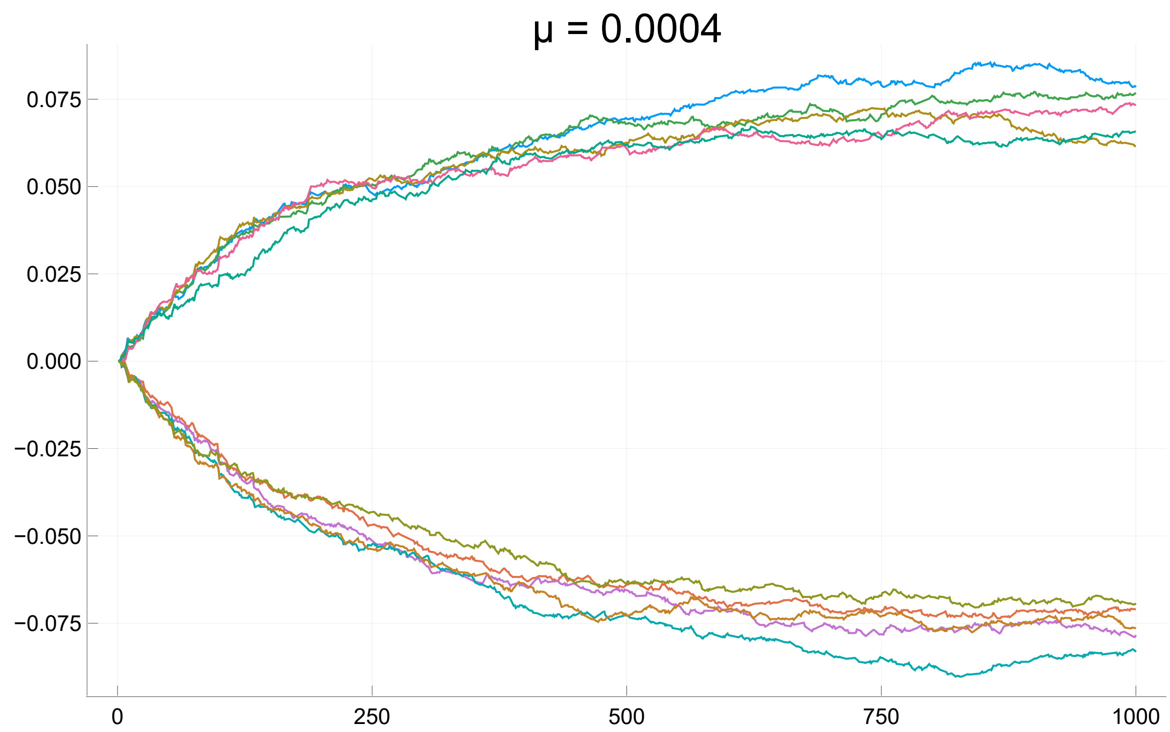


Nos gráficos vemos que os 3 comportamentos possíveis a partir do passo de convergência.

- No primeiro gráfico o valor de μ era muito pequeno e o algoritimo não chegou a convergir para o valor ótimo
- No último gráfico, o valor de μ era muito grande e o algoritimo não consegue convergir para nenhum valor
- No gráfico do meio o algoritimo conseguiu convergir para valores ótimos do filtro, portanto o valor idela deve estar próximo dessa ordem de grandeza

Com isso em mente, vamos alterar o valor de μ manualmente até chegar em um valor ótimo.

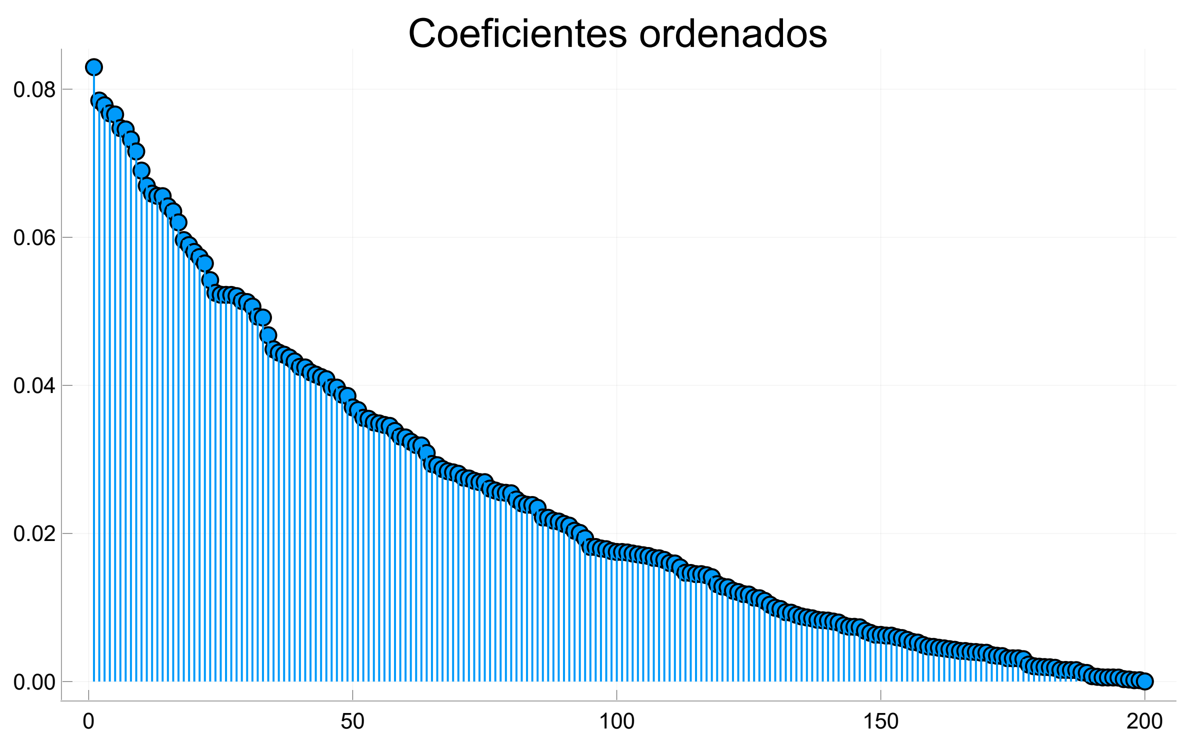
```
begin
    μ = 4e-4

    W_μ_otimo, erro_μ_otimo =  LMS(entrada1, desejado1, M200,N,μ)
    noprint
end
```



Após testes manuais, o valor escolhido para μ foi **μ = 0.0004**, dessa forma o algoritimo tem tempo de convergir e e mantém uma certa estabilidade

Estimação do número de coeficientes

Para estimar o número ideal de coeficientes, vamos plotar o valor final de todos os coeficientes e analisar quantos deles são pequenos e podem ser desconsiderados

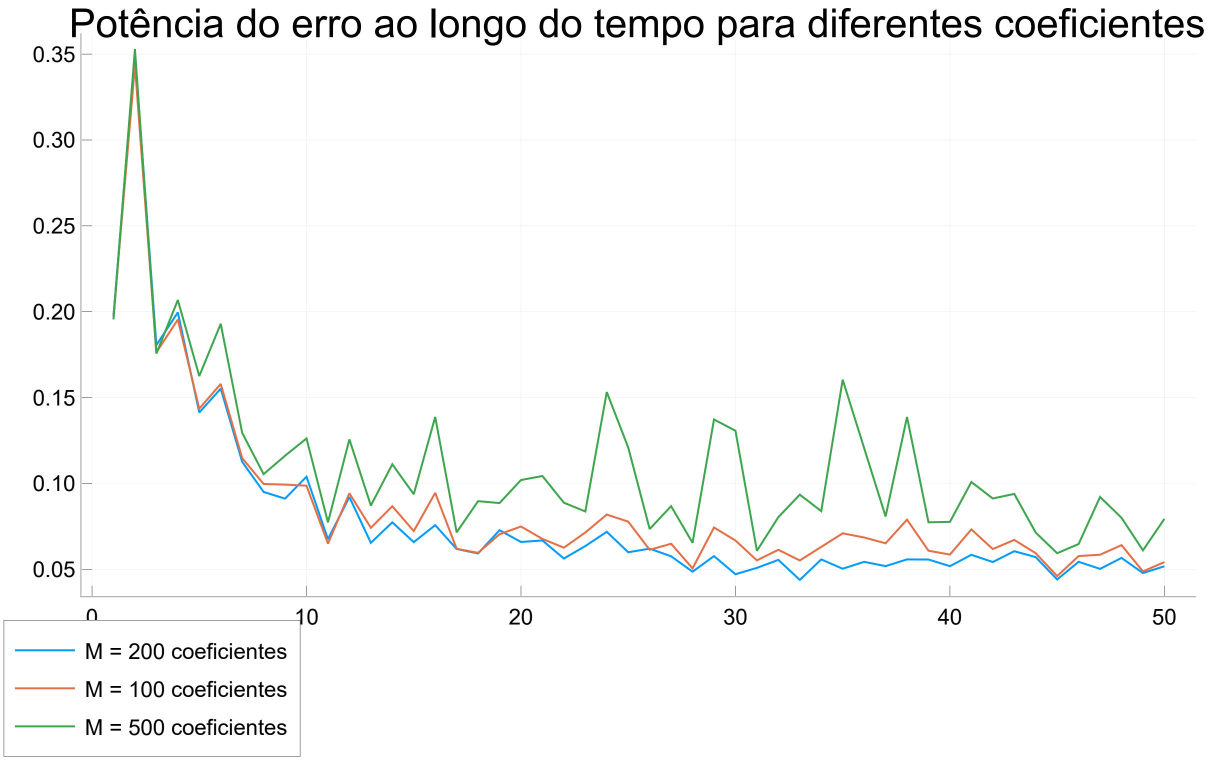


No gráfico vemos que parte dos coeficientes tem um valor bastante baixo e pode ser desconsiderado a fim de diminuir o tamanho do filtro. Os primeiros coeficientes estão em terno de 0.08, portanto iremos desconsiderar os coeficientes uma ordem de grandeza menor (<0.01). Isso nos deixa com algo em torno de **100 coeficientes**.

Para verificar a influência desses coeficientes, vamos plotar a potência do erro em trechos ao longo do tempo dos dois filtros e verificar que a potência dos dois diminui

```
begin
    M = 100
    M50 = 50
    W1, erro1 =  LMS(entrada1, desejado1, M,N,μ)
    W_50, erro_50 =  LMS(entrada1, desejado1, M50,N,μ)
    noprint
end
```

```
begin

    pot_erro = []
    for i in 0:49
        tamanho_trecho = 200
        pot_trecho = sum(erro1[i*tamanho_trecho+1:
(i+1)*tamanho_trecho].^2)/tamanho_trecho
        append!(pot_erro,pot_trecho)
    end

    pot_erro_50 = []
    for i in 0:49
        tamanho_trecho = 200
        pot_trecho = sum(erro_50[i*tamanho_trecho+1:
(i+1)*tamanho_trecho].^2)/tamanho_trecho
        append!(pot_erro_50,pot_trecho)
    end

    pot_erro_200 = []
    for i in 0:49
        tamanho_trecho = 200
        pot_trecho = sum(erro_μ_otimo[i*tamanho_trecho+1:
(i+1)*tamanho_trecho].^2)/tamanho_trecho
        append!(pot_erro_200,pot_trecho)
    end
    noprint
end
```



No gráfico podemos ver a influência do número de coeficientes na saída do filtro.

- Com 200 coeficientes, o gráfico convege para uma potência de erro baixa e constante.
- Com 50 coeficientes, o gráfico chega a convergia para uma potência, mas com mais instabilidade.

Usando **100 coeficientes** como apontado antes, é possível reduzir o tamanho do filtro e ainda ter uma qualidade suficientemente boa para o filtro.

Filtro final

No final dessa etapa o filtro que simula o ambiente real é:

Potência do ruido v[n]

Para calcular a potência do ruído usaremos o fato que nosso filtro se aproxima do sistema real H.

Com isso temos que a saída Y do filtro adaptativo é muito proxima da saída X do sistema real. Portanto:

$$E = Y - D = Y - (V + X) = V$$

portanto:

$$V = Y - D$$

Portanto para calcular a potência do ruído **v[n]** basta passarmos o sinal desejado pelos coeficientes finais do filtro adptativo e calcular sua potência.

```
0.059037475949395936
```

```
begin
    filter = PolynomialRatio(W1[end, :],[1])
    y = filt(filter, entrada1)
    v = y - desejado1
    pot_v = sum(v.^2)/length(v)
end
```

Pot de v[n] = **0.059**

Exercicio 2

```
begin
    fa = 8_000
    desejado2_file = matopen("desejado2.mat");
    desejado2 = read(desejado2_file, "d2")[:,1]
    close(desejado2_file);

    entrada2_file = matopen("entrada2.mat");
    entrada2 = read(entrada2_file, "u2")[:,1]
    close(entrada2_file);
end
```

Entrada

0:00 / 0:23

Eco

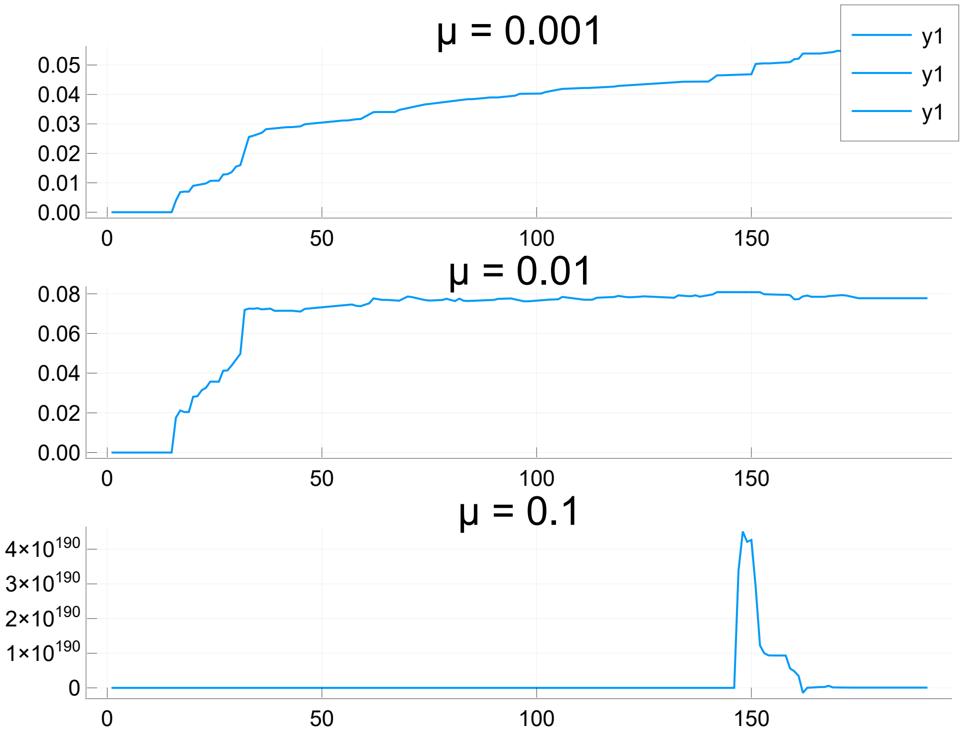
0:00 / 0:23

a) Eliminação do eco.

Para eliminar o eco iremos usar o LMS para tentar convergir o sistema real de que gerou o eco da voz. Com esse sistema estimado, iremos passar o sinal original da voz por ele, e subtrair do sinal com eco. Com isso, esperamos eliminar o eco e ficar apenas com o ruído do som desejado.

Usaremos o mesmo número de coeficientes do ultimo exercício, mas será necessário alterar o passo de adaptação.

Teremos que modificar o passo de adaptação, porque os sinais de entrada mudaram. Isso faz com que o valores de correlação entre sinais que geram o filtro ótimo e consequentemente o valor máximo e ideal de μ também se alteram.

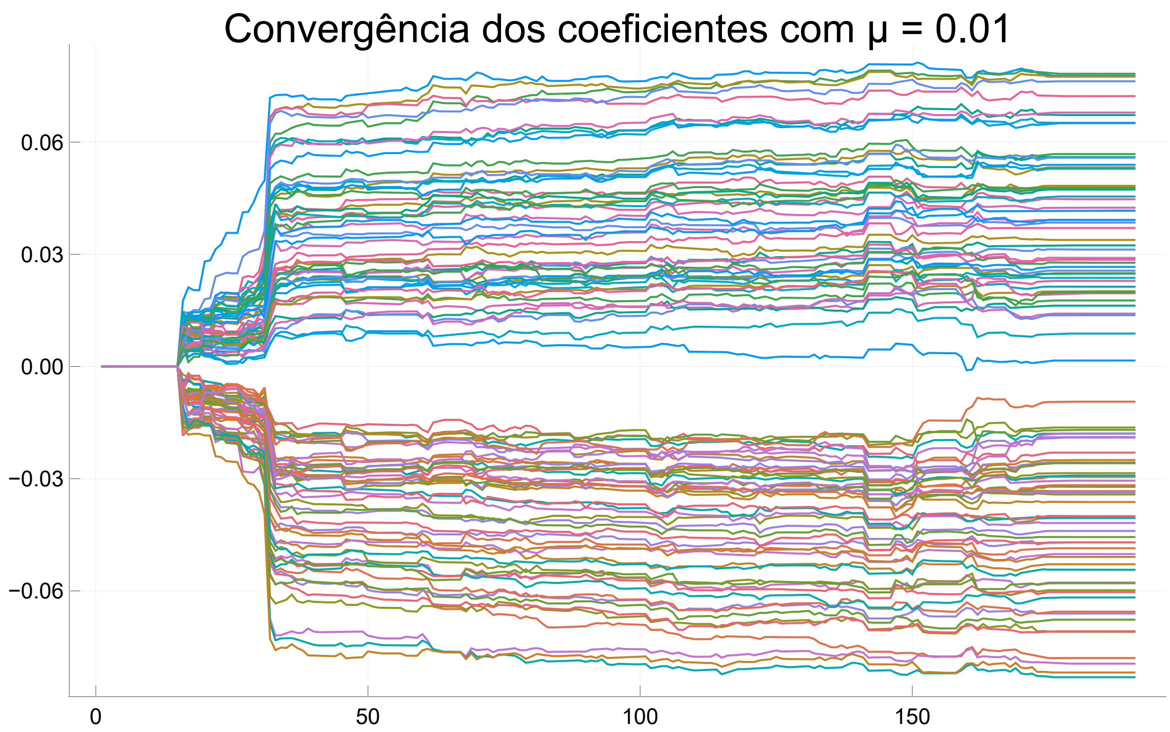


Empiricamente observamos que com

- μ = 0.001, o sistema não tem tempo de convergir
- μ = 0.1, o sistema não converge
- μ = 0.01, o sistema converge bem, portanto o valor ótimo de μ deve estar nessa ordem de grandeza

O μ escolhido foi **μ = 0.01**

```
begin
    N2 = length(entrada2) #190_000 pontors
    μ2 = 1e-2
    W2, erro2 = LMS(entrada2, desejado2, M, N2, μ2)
    noprint
end
```



Da mesma forma que o exercício anterior, se nosso filtro se aproxima do sistema real, temos:

$$X = Y$$

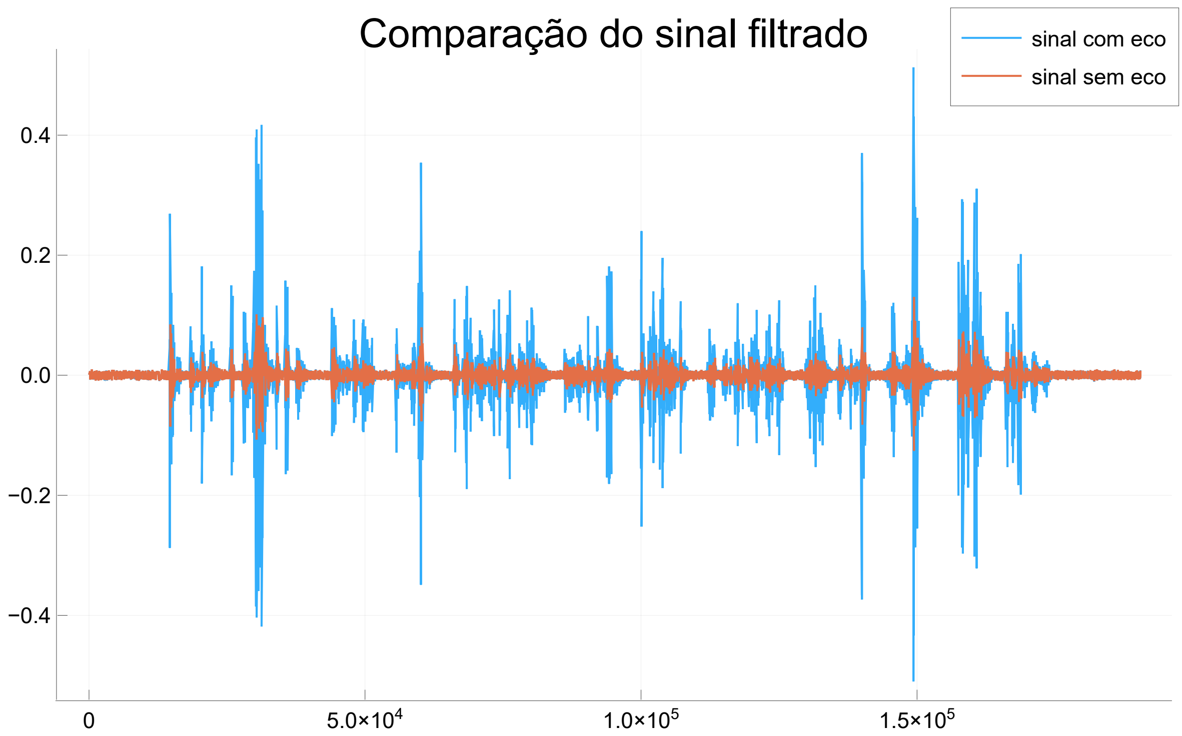
Potanto

$$E = Y - D = Y - (X + V) = V$$

Sendo **X** o sinal de eco e **V** o sinal que queremos obter. Portanto

$$V = Y - D$$

```
begin
    filtro2 = PolynomialRatio(W2[170_000,:], [1])
    y2 = filt(filtro2, entrada2)
    v2 = desejado2 .- y2
    noprint
end
```



b) Potência do ruído de fundo

Após eliminar boa parte do eco, o que sobre é aproximadamente apenas o ruído de fundo, então apenar precisamos calcular a potência do sinal n[n]

```
6.430079651409274e-5
```

```julia
begin
    pot_ruido2 = sum(v2.^2)/length(v2)
end
```

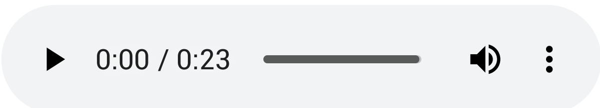
pot ruído = **6.4e-5**

c) Comparação dos sinais

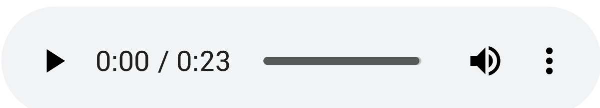
Aqui iremos comparar o sinal de saída. Como o esperado, o sinal ainda possui um pouco da fala do narrador, mas essa fala está bastante atenuada devido a eliminação do filtro adaptativo.

Somado a isso, ouvimo o ruído que calculamos a potência.

Sinal com eco

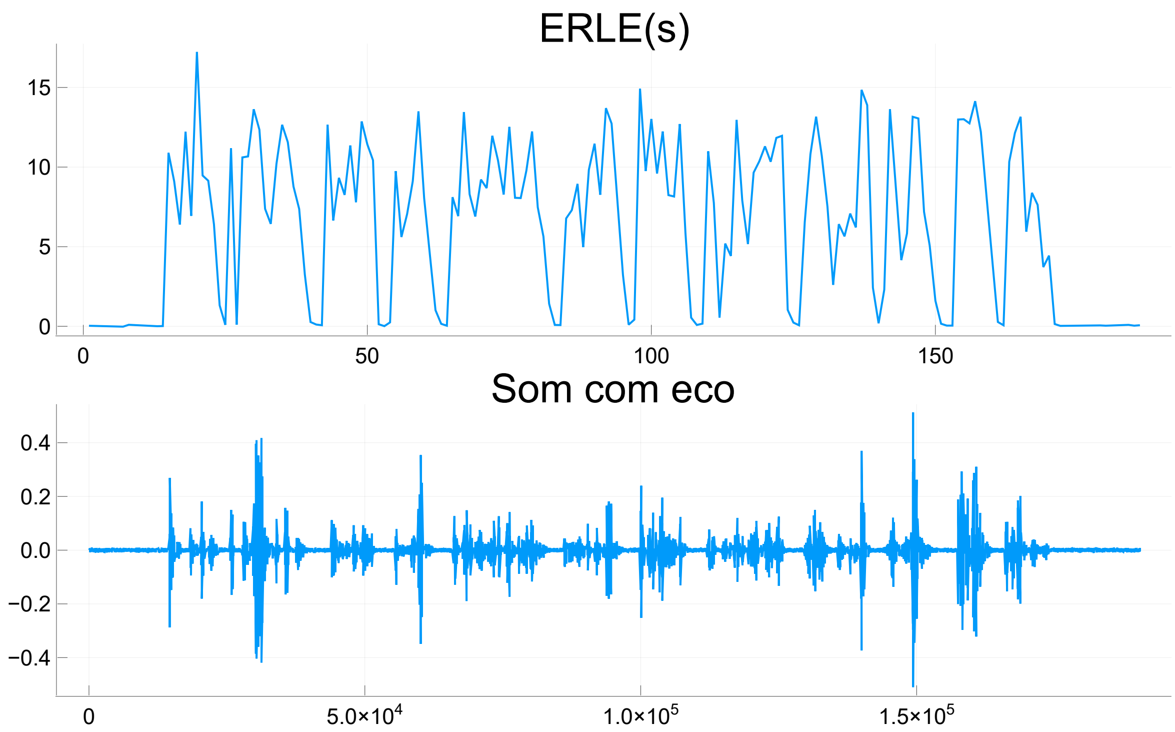


Sinal sem eco

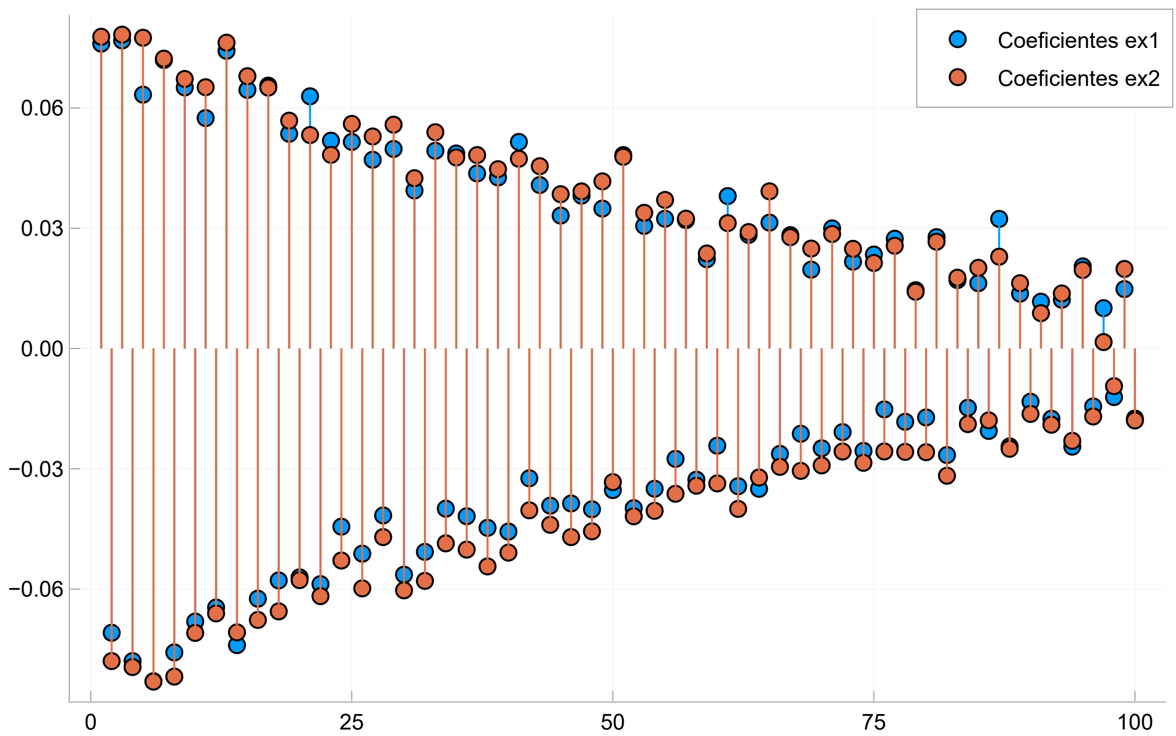


d) Curva ERLE do sinal filtrado

Aqui podemos ver a eficiência do nosso eliminador de eco. No gráfico abaixo vemos a quantidade de eco eliminado em cada trecho de audio. Podemos ver que nos trechos de som, a curva ELRE tem um valor alto indicando que eliminou ruído.



e) Convergência dos coeficientes



Vemos que os coeficientes realmente convergem para algo próximo.

O objetivo desse filtro é simular um sistema real, portanto a razão mais provável do porque os dois filtros se aproximam é que o sistema real que os dois coeficientes tentam simular é o mesmo. Com isso o algoritimo irá fazer com que os coeficientes dos dois sistemas convirjam.

Functions

```
LMS (generic function with 1 method)
```

```
function LMS(x, d, M, N, μ)
    X = zeros(M,1)
    W = zeros(N,M)
    erro = zeros(N,1)

    for n in 1:N-1
        X = [x[n]; X[1:M-1]]
        y = W[n,:]'*X
        erro[n] = d[n]-y
        W[n+1, :] = W[n,:] +μ*erro[n]*X
    end
    return W, erro
end
```

```
pow2db (generic function with 1 method)
```

```
function pow2db(x)
    20*log10(x)
end
```

```
noprint =
```

```
noprint = md""
```

ERLE (generic function with 1 method)

```
function ERLE(d,e,Nw,fa)
# %
# % Função para cálcudo do ERLE
# % (echo return loss enhancement)
# % após o cancelador de eco
# %
# % Saída
# % Estimativa do ERLE em dB
# %
# % Entradas
# % d: sinal de eco (desejado)
# % e: sinal de eco residual (erro do filtro adaptativo)
# % Nw: número de amostras na janela para estimativa do eco ao longo do tempo
# % fa: frequência de amostragem
# %
# %

# % MTMS, 08/2018

N= length(d);
Nb= (Int)(floor(N/Nw));
ERLEx = zeros(1,Nb);
Ta= 1/fa ;

for i in 1:Nb
    l=Nw*(i-1)+1:Nw*i;
    ERLEx[i] = mean(d[l].^2)/mean((e[l].+eps()).^2);
end
ERLEdB=10*log10.(ERLEx);

return ERLEdB[1,:]

# figure
# t=0:(N-1)*Ta/(N/Nw-1):(N-1)*Ta;
# plot(t,ERLEdB)
# grid on
# hold on
# ylabel('     Eco                                      ERLE (dB)')
# xlabel('tempo (s)')
# td=t(1):(t(end)-t(1))/(length(d)-1):t(end);
# plot(td,10*d/max(abs(d))-11)
# plot([td(1) td(end)],[-0.5 -0.5],'k','LineWidth',1)
# set(gca,'YTick',[0:10:max(ERLEdB), ceil(max(ERLEdB))]');
# axis([td(1) td(end) -21 ceil(max(ERLEdB))])
end
```

```
Enter cell code...
```