EXP 3

- Gabriel Tavares - 10773801
- Guilherme Reis - 10773700


```
PlotlyBackend()
```

```
begin
    using DSP
    using Plots
    using LinearAlgebra
    plotly()
end
```

1) Simulação do filtro de 2 coeficientes

Parâmetros de simulação

```
begin
    N = 200
    realizacoes = 100
    σₓ² = 1
    σᵥ² = 0.01
    noprint
end
```

Simulação

```
begin
    M = 2
    w_lms1 = zeros(N, M)
    erro1 = zeros(N)

    h1 = [1, -0.5]
    μ = 0.05
    for realizacao in 1:realizacoes
        x = randn(N)*sqrt(σₓ²)
        v = randn(N)*sqrt(σᵥ²)
        y = filt(h1, x)
        d = y .+ v
        w, erro = LMS(x, d, M, μ)
        w_lms1 += w
        erro1 += erro.*erro
    end

    w_lms1 = w_lms1./realizacoes
    erro1 =  erro1./realizacoes
    noprint
end
```

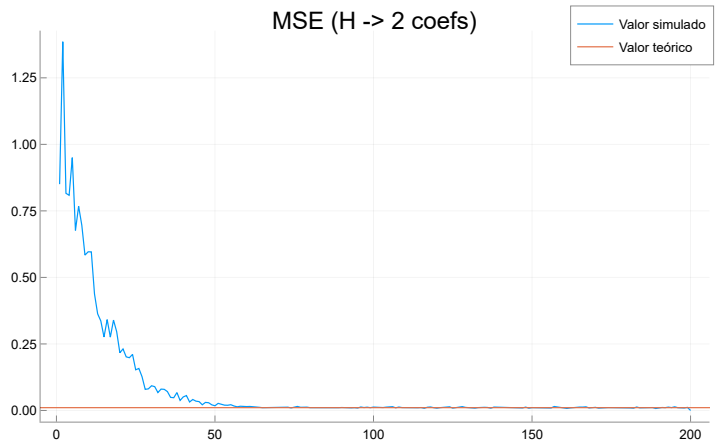
Valores teóricos

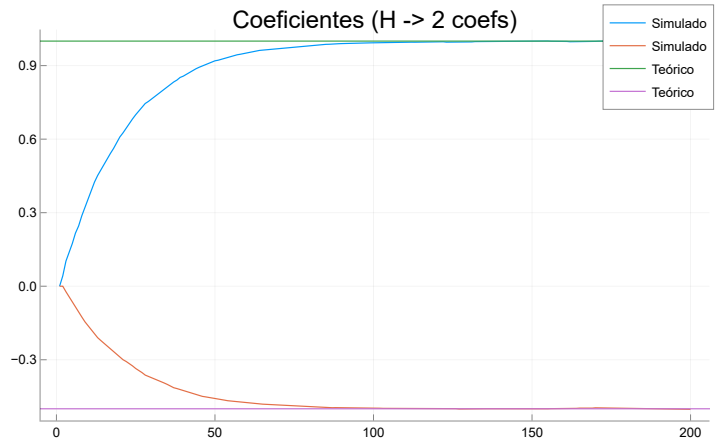
```
begin
    Rϕ1 = [σₓ² 0;
           0 σₓ²]
    rdϕ1 = [h1[1]*σₓ², h1[2]*σₓ²]
    K1 = [μ*σᵥ²/2 0;
          0 μ*σᵥ²/2]
    wₒ1 = inv(Rϕ1)*rdϕ1

    EMSE = tr(Rϕ1*K1)

    noprint
end
```

Comparação de valores simulados e teóricos





Como esperado, os valores simulados e teóricos são bastante próximos. Como o filtro ideal e LMS tem 2 coeficientes, os coeficientes finais tendem para o filtro ideal

2) Simulação com filtros de 1 e 3 coeficientes

Simulação 1 coeficiente

```
begin
    # M = 2
    w_lms2 = zeros(N, M)
    erro2 = zeros(N)

    h2 = [1]
    # μ = 0.1
    for realizacao in 1:realizacoes
        x = randn(N)*sqrt(σₓ²)
        v = randn(N)*sqrt(σᵥ²)
        y = filt(h2, x)
        d = y .+ v
        w, erro = LMS(x, d, M, μ)
        w_lms2 += w
        erro2 += erro.*erro
    end

    w_lms2 = w_lms2./realizacoes
    erro2 =  erro2./realizacoes
    noprint
end
```

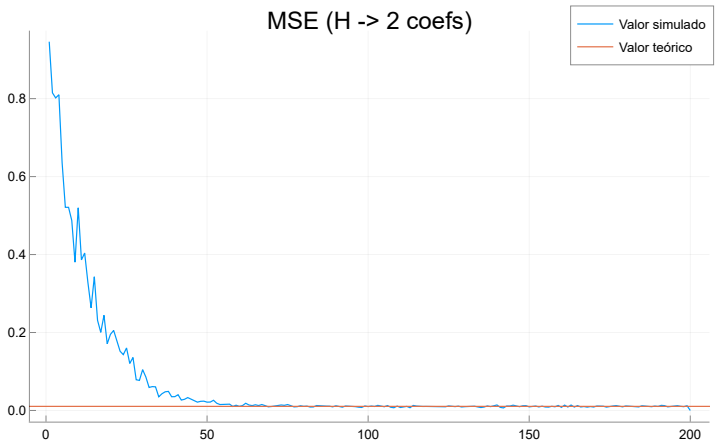
Valores teóricos

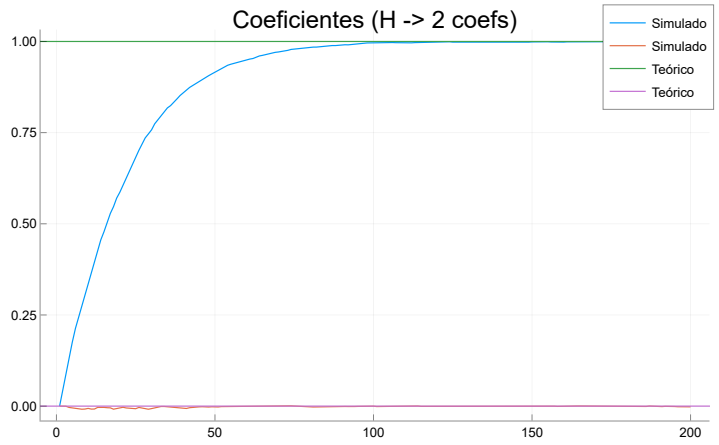
```
begin
    Rϕ2 = [σₓ² 0;
        0 σₓ²]
    rdϕ2 = [h2[1]*σₓ², 0]
    K2 = [μ*σᵥ²/2 0;
        0 μ*σᵥ²/2]
    w₀2 = inv(Rϕ2)*rdϕ2

    EMSE2 = tr(Rϕ2*K2)

    noprint
end
```

Comparação de valores simulados e teóricos





Simulação 3 coeficientes

```
begin
    # M = 2
    w_lms3 = zeros(N, M)
    erro3 = zeros(N)

    h3 = [1, -0.5, 0.2]
    # μ = 0.1
    for realizacao in 1:realizacoes
        x = randn(N)*sqrt(σₓ²)
        v = randn(N)*sqrt(σᵥ²)
        y = filt(h3, x)
        d = y .+ v
        w, erro = LMS(x, d, M, μ)
        w_lms3 += w
        erro3 += erro.*erro
    end

    w_lms3 = w_lms3./realizacoes
    erro3 =  erro3./realizacoes
    noprint
end
```

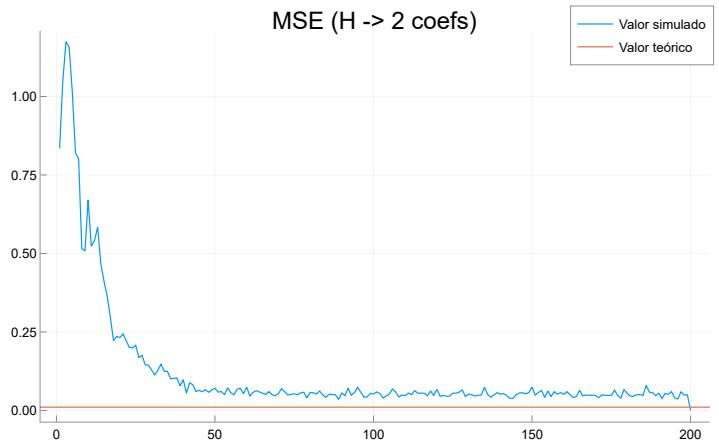
Valores teóricos

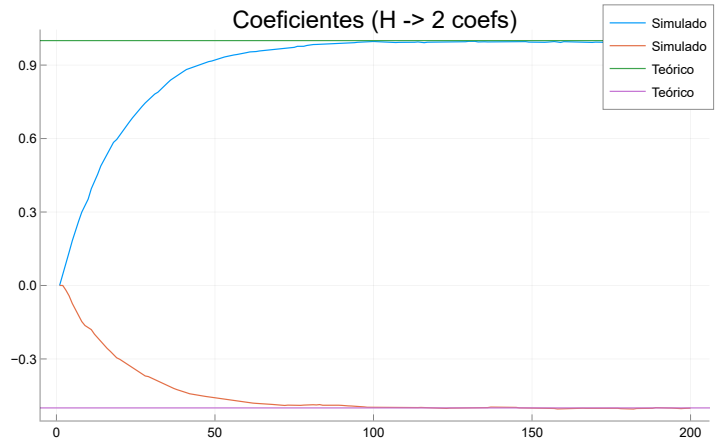
```
begin
    Rϕ3 = [σₓ² 0;
        0 σₓ²]
    rdϕ3 = [h3[1]*σₓ², h3[2]*σₓ²]
    K3 = [μ*σᵥ²/2 0;
        0 μ*σᵥ²/2]
    wₒ3 = inv(Rϕ2)*rdϕ3

    EMSE3 = tr(Rϕ2*K2)

    noprint
end
```

Comparação de valores simulados e teóricos





Usando um filtro ideal de 2 e 3 coeficientes, vemos que no primeiro caso no o algoritmo converge pra um resultado bom e se mantém na faixa de valores teóricos de EMSE. O primeiro coeficiente do filtro tende ao coeficiente do filtro ideal.

Com 3 coeficientes, o filtro LMS tenta chegar no valor mais próximo possível do filtro real, mas tem uma limitação de projeto de atingir o filtro ideal. Devido a iddo, o seu valor de erro não na convergência fica a cima do valor teórico

3) Simulação do filtro adaptativo de 5 coeficientes e o filtro real de 2

Simulação

```
begin
    M5 = 5
    w_lms4 = zeros(N, M5)
    erro4 = zeros(N)

    h4 = [1, -0.5]
    # μ = 0.1
    for realizacao in 1:realizacoes
        x = randn(N)*sqrt(σₓ²)
        v = randn(N)*sqrt(σᵥ²)
        y = filt(h4, x)
        d = y .+ v
        w, erro = LMS(x, d, M5, μ)
        w_lms4 += w
        erro4 += erro.*erro
    end

    w_lms4 = w_lms4./realizacoes
    erro4 =  erro4./realizacoes
    noprint
end
```

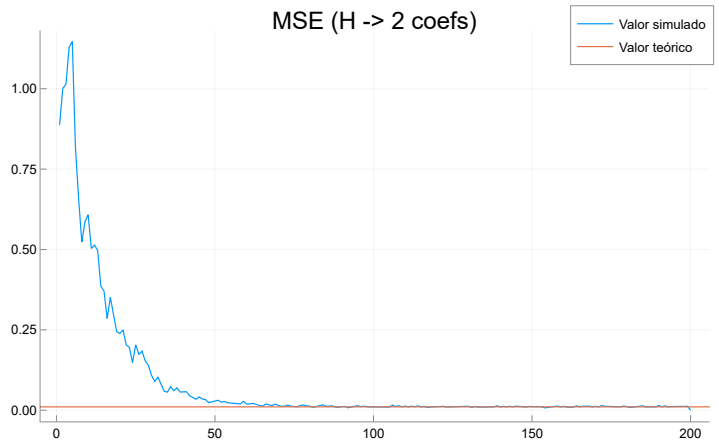
Valores teóricos

```
begin
    Rϕ5 = [σₓ² 0 0 0 0;
        0 σₓ² 0 0 0;
        0 0 σₓ² 0 0;
        0 0 0 σₓ² 0 ;
        0 0 0 0 σₓ²;
    ]
    rdϕ5 = [h4[1]*σₓ², h4[2]*σₓ², 0 ,0, 0]
    K5 = [μ*σᵥ²/2 0;
        0 μ*σᵥ²/2]
    w₀5 = inv(Rϕ5)*rdϕ5

    EMSE5 = tr(Rϕ2*K5)

    noprint
end
```

Comparação de valores simulados e teóricos



UndefVarError: w₀4 not defined

1. **top-level scope** @ *Local: 4*

```
begin
    plot(title= "Coeficientes (H -> 2 coefs)")
    plot!(w_lms4, label="Simulado")
    hline!([w₀4[1]], label="Teórico")
    hline!([w₀4[2]], label="Teórico")
end
```

Nesse exemplo, como esperado o filtro LMS consegue convergir para os valores ideiais e teóricos, por ter coeficientes suficiente para representar o filtro

4) Filtrando x[n] antes da adaptação

Aqui estamos usando sinal de entrada x[n] mais complexo do que apenas um ruído gaussiano

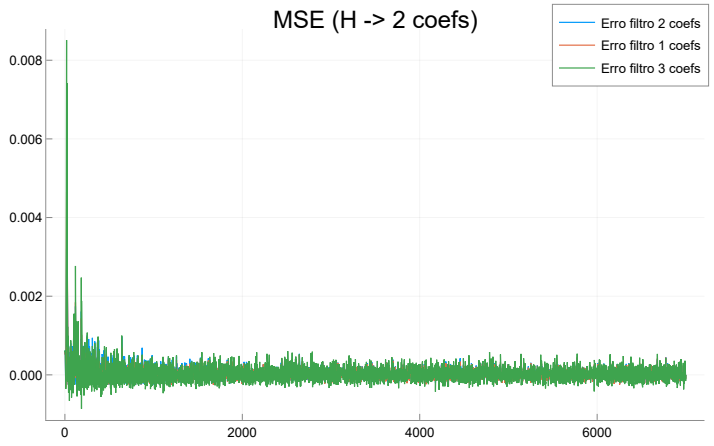
Simulação

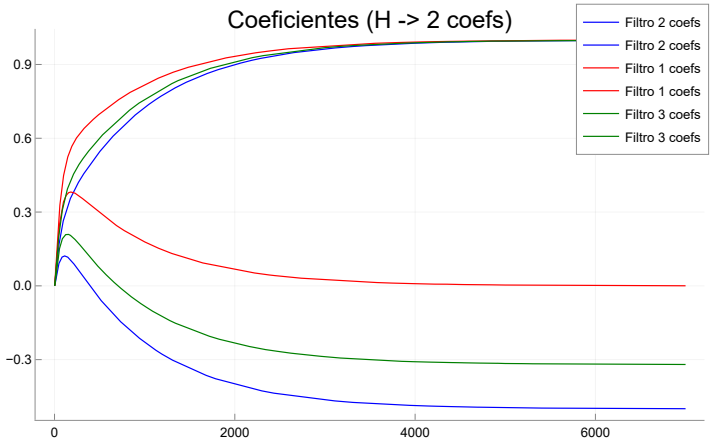
```
begin
    N2 = 7_000
    w_lms5 = zeros(N2, M)
    w_lms6 = zeros(N2, M)
    w_lms7 = zeros(N2, M)
    erro5 = zeros(N2)
    erro6 = zeros(N2)
    erro7 = zeros(N2)

    realizacoes2 = 1_000

    σᵤ² = 1
    h5 = [1, -0.5]
    h6 = [1]
    h7 = [1, -0.5, 0.2]
    a = 0.9
    g = PolynomialRatio([sqrt(1-a^2)],[1, -a] )
    μ2 = 0.01
    for realizacao in 1:realizacoes2
        global w_lms5
        global w_lms6
        global w_lms7
        global erro5
        global erro6
        global erro7
        u = randn(N2)*sqrt(σᵤ²)
        x = filt(g, u)
        v = randn(N2)*sqrt(σᵥ²)
        y5 = filt(h5, x)
        y6 = filt(h6, x)
        y7 = filt(h7, x)
        d5 = y5 .+ v
        d6 = y6 .+ v
        d7 = y7 .+ v
        w5, erro5 = LMS(x, d5, M, μ2)
        w6, erro6 = LMS(x, d6, M, μ2)
        w7, erro7 = LMS(x, d7, M, μ2)
        w_lms5 += w5
        w_lms6 += w6
        w_lms7 += w7
        erro5 += erro5.*erro5
        erro6 += erro6.*erro7
        erro7 += erro7.*erro6
    end

    w_lms5 = w_lms5./realizacoes2
    w_lms6 = w_lms6./realizacoes2
    w_lms7 = w_lms7./realizacoes2
    erro5 =  erro5./realizacoes2
    erro6 =  erro6./realizacoes2
    erro7 =  erro7./realizacoes2
    noprint
end
```





Nesse exemplo conseguimos ver que os valores simulados ainda tendem para os mesmos valores obtidos anteriormente, mas por ter um sinal mais complexo que um ruído puro, o tempo de convergência é bem maior

Functions

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

```
function LMS(x, d, M, μ)
    N = length(x)
    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
```

```
noprint =
```

```
noprint = md""
```