ExperiÊncia 06

Gabriel Tavares - 10773801

Guilherme Reis - 10773700

```
PlotlyBackend()
```

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

Arranjos de antenas e ganho espacial

01 - Projeto de antena DAS

Projeto de arranjo de antena DAS miradas para 20º usando os coeficientes de cada antena definida por:

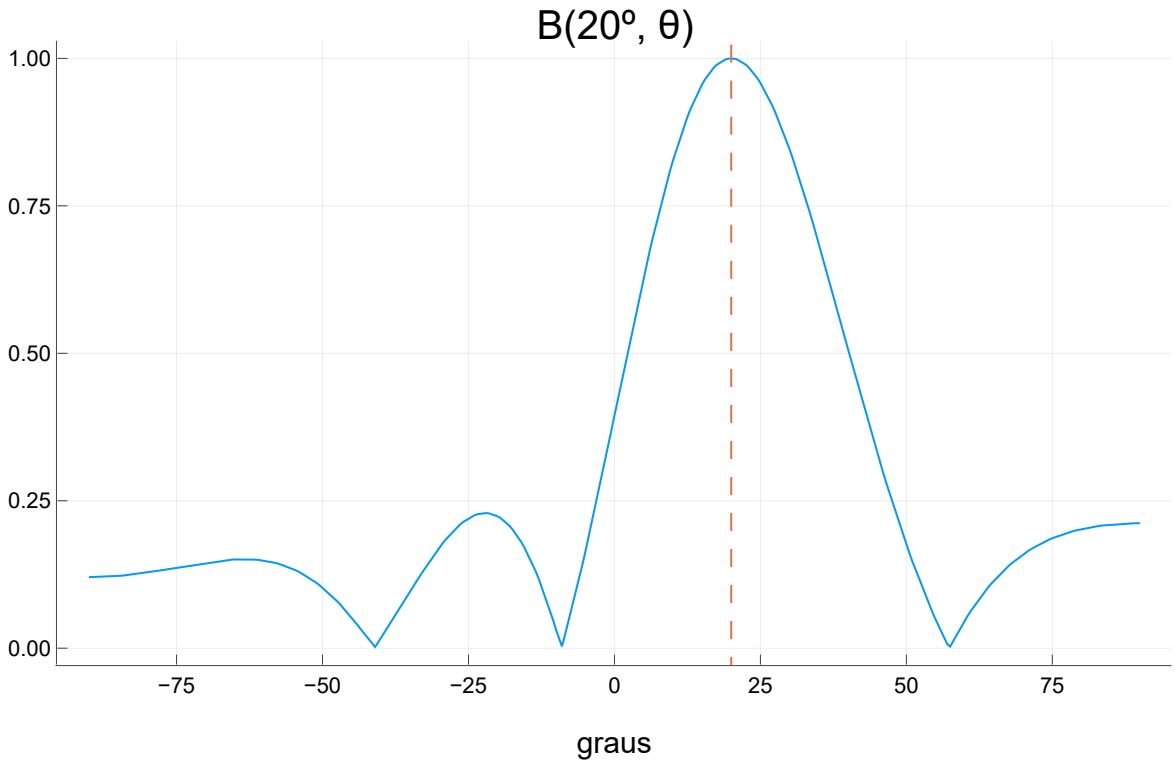
$$a_m = \frac{1}{M} e^{j\Omega\tau(\theta)}$$

E com os coeficientes das antenas definimos o ganho espacial como

$$B(\theta) = \sum a_m e^{-j\Omega\tau(\theta)}$$

```
begin
    M = 8
    F = 60e9 #Hz
    Ω = 2*π*F
    c = 3e8 #m/s
    λ = c/F #m
    j = im
    θ0 =  20 #graus
    noprint
end
```

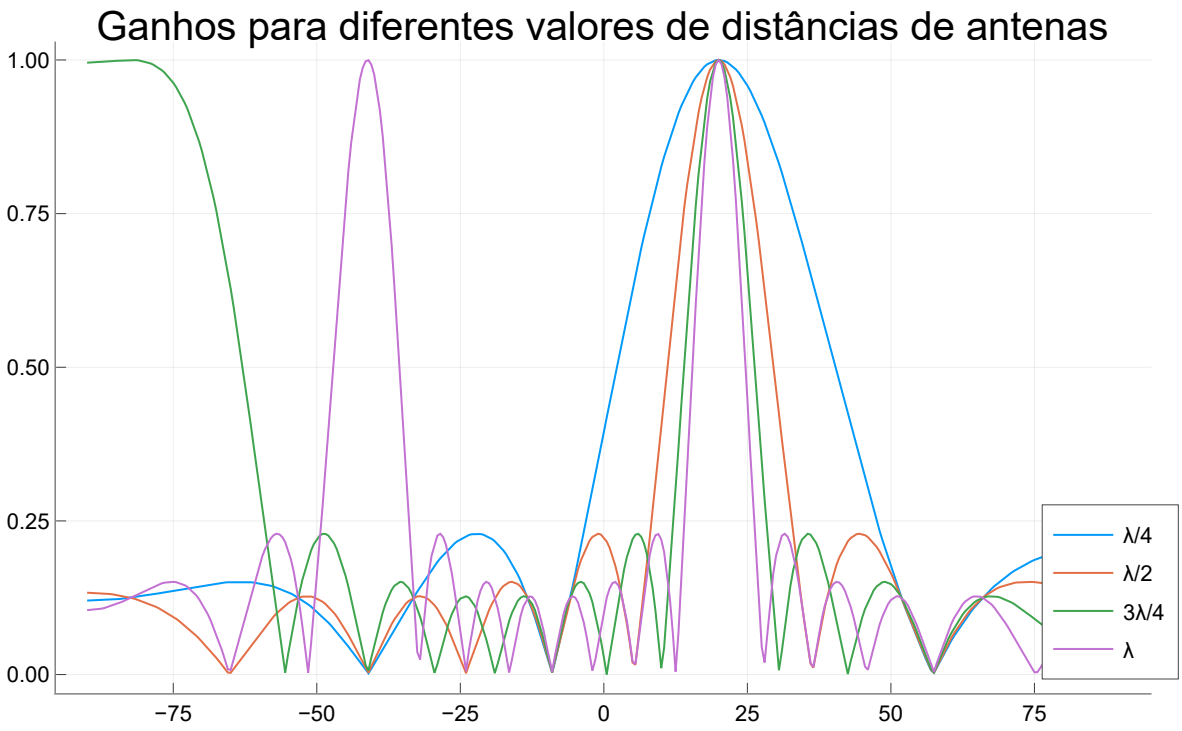
```
begin
    a = DAS(00, M, λ/4)
    θ = -90:0.5:90
    B20 = B(a, θ, λ/4)
    noprint
end
```



02 - Diferentes ganhos

Vamos calcular o ganho das antenas para diferentes valores de distâncias.

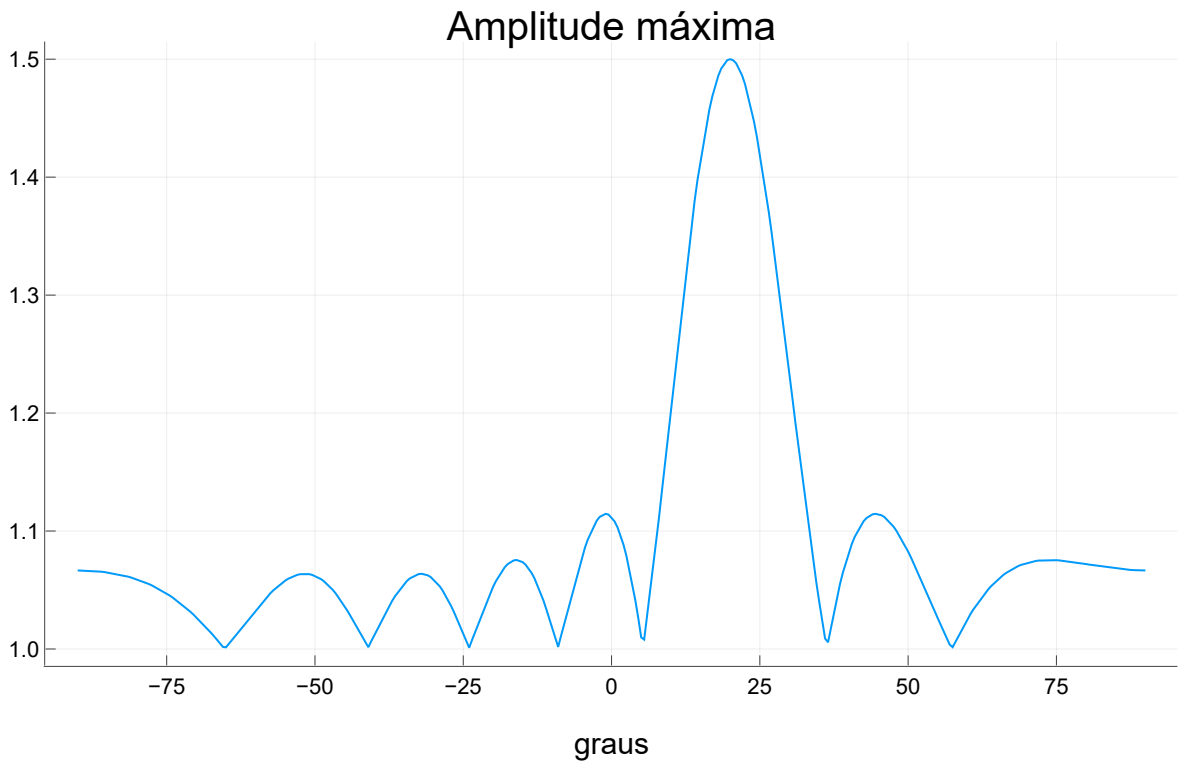
Distâncias : λ/4, λ/2, 3λ/4, λ



03 - Ganho da interferência

Dado um sinal x1 vindo de θ1 conhecido e um sinal x2 vindo de θ2, queremos saber a amplitude máxima do sinal a depender da direção θ2 da interferência

```
begin
    A1=1
    A2=0.5
    a3 = DAS(20, M, λ/2)
    A_ =  A1*abs(B(a3, 20, λ/2)) .+ A2.*abs.(B(a3, θ, λ/2))
    noprint
end
```



Sinal Real

```
begin
    fa = 1e12
    A0 = 3
    θ1 = 45 #direção da primeira interferência
    θ2 = -15 #direção da segunda interferência
    x = matread("sinais.mat")["sinal_recebido"] # sinais de cada antenas
    noprint
end
```

1 - Processamento DAS do sinal

Filtro Passa-Baixas

Implementação do filtro Passa-Biaxas com frequÊncia de corte igual a portadora.

Usaremos um filtro Butterworth de 6 coeficientes.

```
begin
    fc = 2*F/fa
    pb = digitalfilter(Lowpass(F, fs = fa) ,Butterworth(6))
    PB, ω = freqresp(pb)
    noprint
end
```


```
begin
    plot(ω, abs.(PB), label =false)
    plot!(title= "Resposta do filtro", xlabel = "rad/amostra")
    vline!([fc*π], label = "Freq corte")
end
```

Divisão dos sinais em componente imaginária e real

Nessa etapa multiplicamos o sinal por $2cos(\Omega)$ e $-2sen(\Omega)$ para reinterpretar o sinal real como uma parte imaginária e uma parte real do sinal modulado.

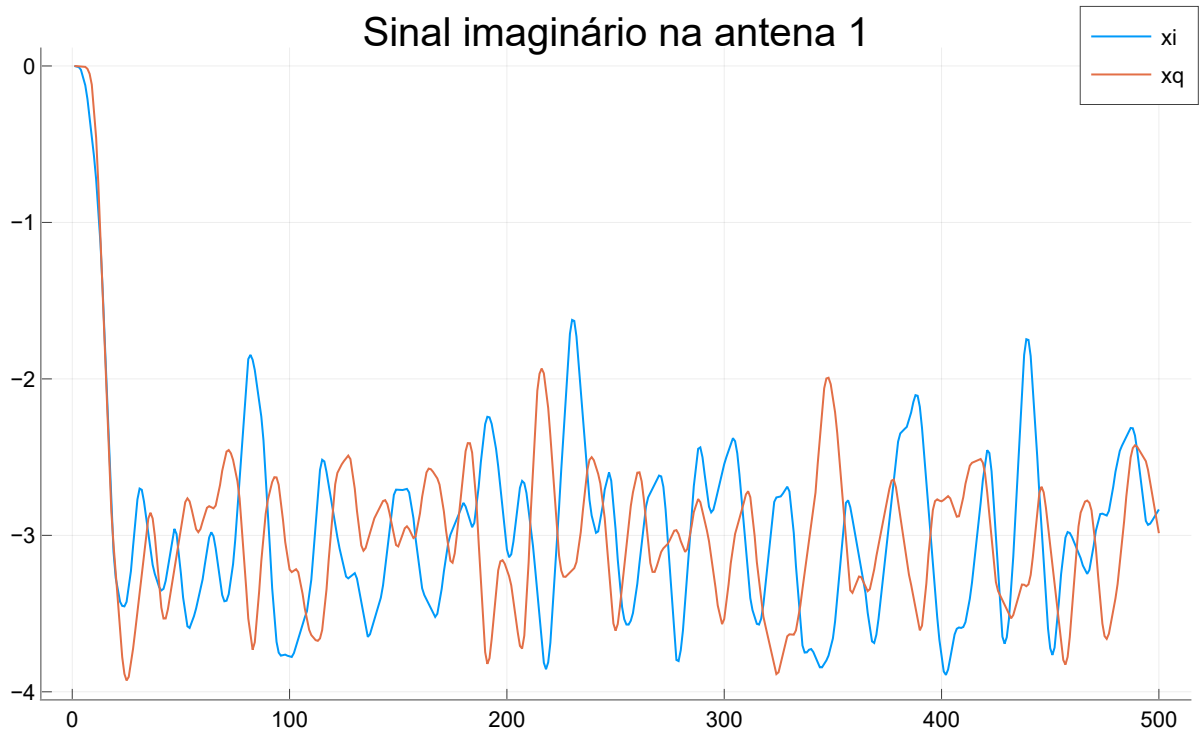
```
begin
    coss = 2*cos.(Ω* range(0, length = size(x)[1], step = 1/fa))
    senn = -2*sin.(Ω* range(0, length = size(x)[1], step = 1/fa))


    x̃i = zeros(size(x))
    x̃q = zeros(size(x))

    for m in 1:M
        x̃i[:, m] = x[:,m].*coss
        x̃q[:, m] = x[:,m].*senn
    end

    xi = zeros(size(x))
    xq = zeros(size(x))

    for m in 1:M
        xi[:, m] = filt(pb, x̃i[:,m])
        xq[:, m] = filt(pb, x̃q[:,m])
    end
end
```



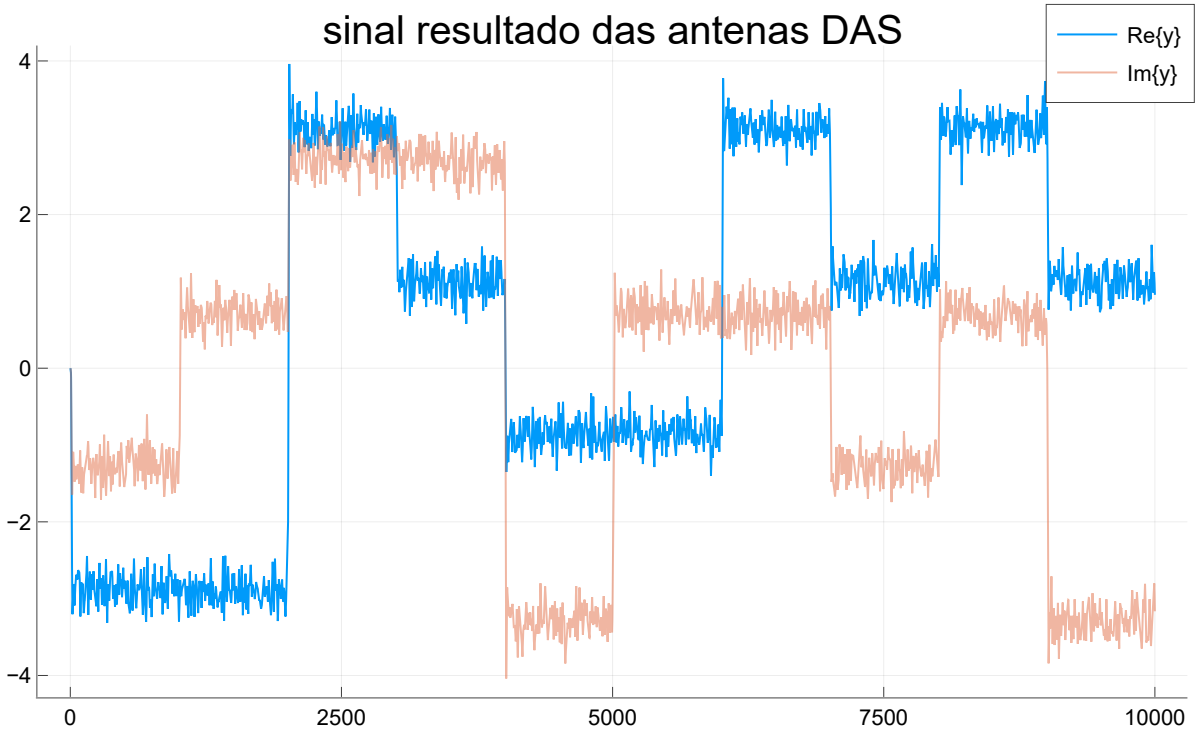

```
begin
    plot(xi[:,1][1:500], label = "xi")
    plot!(xq[:,1][1:500], label = "xq")
    plot!(title = "Sinal imaginário na antena 1")

end
```

2 - Projeto dos coeficientes das antenas

Tendo o sinal reinterpretado em cada antena, vamos multiplicar cada sinal por um coeficiente definido da mesma maneira que o item anterior

```
begin
    w = DAS(θ0, M, λ/2)
    xim = xi + j*xq
    y = xim * conj.(w)
    noprint
end
```



3 - Decodificação

A decodificação irá determinar o tamanho do intervalo de transmissão de cada símbolo calcula o valor médio do número nesse intervalo e aproxima pra algum valor da tabela de 16QAM.

```
begin
    N = round(Int, fa*1e-9)

    codigo = QAM(mean(real.(y[1:N]))+ j* mean(imag.(y[1:N])))

    noprint
end
```

O simbolo do primeiro intervalo lido é **3**

4 - Decodificação do sinal completo

Agora repetimos o processo a cima em todos os 10 trechos de N amostras

```
[3.0, 1.0, 8.0, 12.0, 6.0, 5.0, 9.0, 15.0, 9.0, 14.0]
```

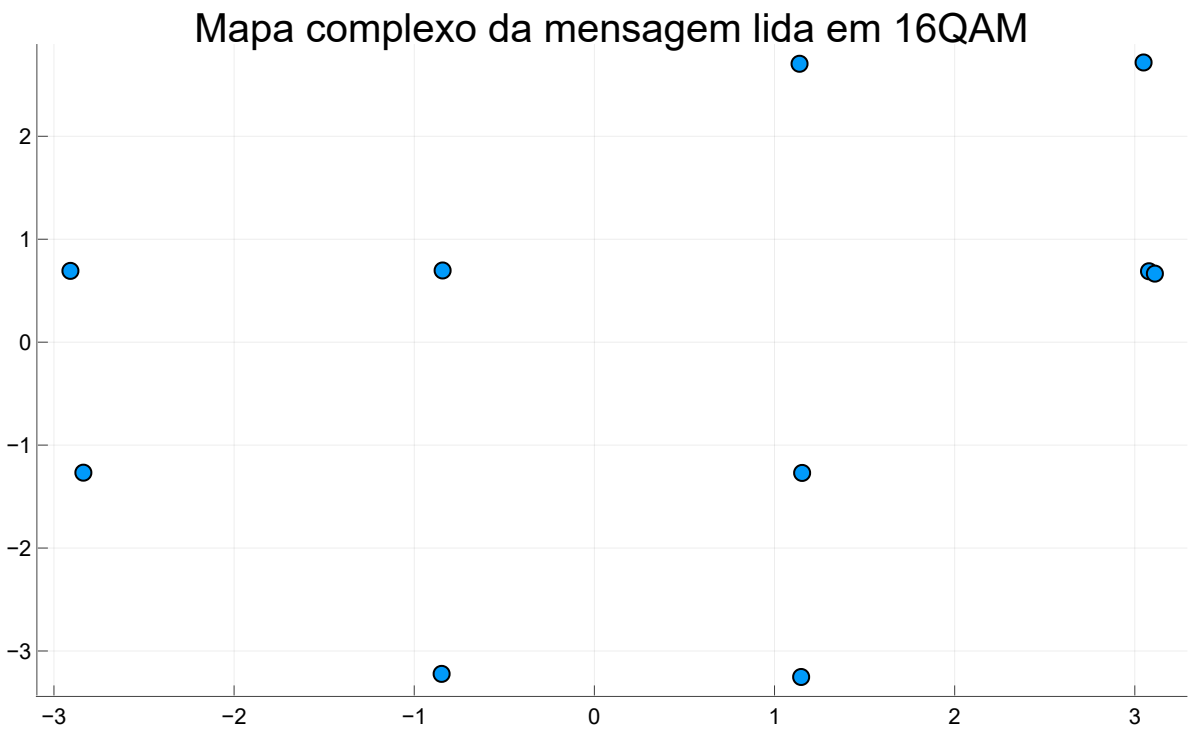
```
begin
    mensagem = matread("mensagem.mat")["mensagem"]

    num_simbolos = round(Int,size(x)[1]/N)
    mensagem_lida = zeros(num_simbolos)
    QAM_lido = zeros(num_simbolos,2)

    for i in 1:num_simbolos
        QAM_lido[i,1] = mean(real.(y[(i-1)*N + 1 : (i)*N]))
        QAM_lido[i,2] = mean(imag.(y[(i-1)*N + 1 : (i)*N]))

        mensagem_lida[i] = QAM(QAM_lido[i,1]+ j* QAM_lido[i,2])
    end

    mensagem_lida
end
```



05 - Adição de ruído ao sinal

Depois iremos repetir o processo a cima de leitura das antenas mas adicionando um ruído gaussiano nos sinais de entrada para verificar quão robusto é a modulação a ruidos.

```
begin
    A_ruido = 30 #Amplitude do ruído
    x_ruido = x + A_ruido*randn(size(x))

    x̃i_ruido = zeros(size(x))
    x̃q_ruido = zeros(size(x))

    for m in 1:M
        x̃i_ruido[:, m] = x_ruido[:,m].*coss
        x̃q_ruido[:, m] = x_ruido[:,m].*senn
    end

    xi_ruido = zeros(size(x))
    xq_ruido = zeros(size(x))

    for m in 1:M
        xi_ruido[:, m] = filt(pb, x̃i_ruido[:,m])
        xq_ruido[:, m] = filt(pb, x̃q_ruido[:,m])
    end

    # w = DAS(θ0, M, λ/2)
    xim_ruido = xi_ruido + j*xq_ruido
    y_ruido = xim_ruido * conj.(w)

    noprint
end
```

```
[2.0, 1.0, 8.0, 8.0, 6.0, 5.0, 9.0, 11.0, 9.0, 14.0]
```

```
begin
    mensagem_lida_ruido = zeros(num_simbolos)
    QAM_lido_ruido = zeros(num_simbolos,2)

    for i in 1:num_simbolos
        QAM_lido_ruido[i,1] = round(mean(real.(y_ruido[(i-1)*N + 1 : (i)*N])))
        QAM_lido_ruido[i,2] = round(mean(imag.(y_ruido[(i-1)*N + 1 : (i)*N])))

        mensagem_lida_ruido[i] = QAM(QAM_lido_ruido[i,1] + j*QAM_lido_ruido[i,2])
    end

    mensagem_lida_ruido
end
```

```
10×1 Matrix{Float64}:
 -1.0
  0.0
  0.0
 -4.0
  0.0
  0.0
  0.0
 -4.0
  0.0
  0.0
```

```
mensagem_lida_ruido - mensagem
```

Adicionando um ruído de amplitude 30 vemos que ainda temos uma taxa de acerto bastante alta. Isso ocorre porque a média de um ruído tende a zero quando temos pontos suficientes

Functions

Definições das funções usadas nesse código

```
noprint =
```

```
noprint = md""
```

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

```
function τ(θ, m, d; c = 3e8)
    return m*d*sind(θ)/c
end
```

```
B (generic function with 3 methods)
```

```
function B(a, θ, d ;c = 3e8)
    M = length(a)
    B_ = 0 + 0*j

    for m in 1:M
        B_ += a[m]*exp(-j*Ω*τ(θ,m, d; c = c))
    end
    return B_
end
```

```
B (generic function with 2 methods)
```

```
function B(a, θ::Vector{Float64}, d ;c = 3e8)

    N = length(θ) #numero de graus do ganho
    B_ = zeros(Complex, N)

    for i in 1:N
        B_[i] = B(a, θ[i], d; c = c)
    end

    return B_
end
```

```
B (generic function with 3 methods)
```

```
function B(a, θ::StepRangeLen{Float64}, d; c = 3e8)
    return B(a, collect(θ),d; c=c)
end
```

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

```
function DAS(θ, M, d; Ω = Ω, c=3e8)
    a = zeros(Complex , M )
    for m in 1:M
        a[m] = 1/M * exp(j * Ω*τ(θ, m-1, d; c=c))
    end
    return a
end
```

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

```
function QAM(numero::Complex{})

    QAM_table = [
        -3+3j, # 0
        -3+1j, # 1
        -3-3j, # 2
        -3-1j, # 3
        -1+3j, # 4
        -1+1j, # 5
        -1-3j, # 6
        -1-1j, # 7
        3+3j,  # 8
        3+1j,  # 9
        3-3j,  # 10
        3-1j,  # 11
        1+3j,  # 12
        1+1j,  # 13
        1-3j,  # 14
        1-1j   # 15
        ]

    min_dist = Inf
    dist = Inf
    melhor_indice = 0

    for indice in 1:16
        dist = abs(numero - QAM_table[indice])
        if dist < min_dist
            min_dist = dist
            melhor_indice = indice-1
        end
    end

    return melhor_indice
end
```