Relatório Projeto 1

Comunicações Móveis

Autoria	**Matrícula**
Pedro Henrique Dornelas Almeida	242110048

Programa de Pós Graduação em Engenharia Elétrica
Universidade de Brasília

6 de outubro de 2025

1 Simulação

O projeto foi desenvolvido em Matlab e pode-se dividir o código em três partes principais:

1. Definição de Variáveis: faz-se a definição de variáveis globais que serão utilizadas ao longo do programa;

2. Simulação e figuras: faz-se toda a simulação e lógicas necessárias para implementar as fórmulas e geração de variáveis aleatórias tornando possível a geração das figuras;

3. Funções Auxiliares: no trecho final do código estão as funções auxiliares que serão utilizadas durante o código, em geral são funções criadas para facilitar o entendimento e encapsular uma lógica específica. Pode ser reaproveitável em vários locais, o que facilita a reprodução, caso necessário.

A simulação foi realizada em um único arquivo e segue a ordem descrita no guia do projeto e nos slides apresentados em sala. O cenário sorteado e utilizado foi UMa-NLoS - Macro celular urbano e sem visada direta.

Para rodar a simulação, basta executar o arquivo *simulation.m*. Disponível em https://github.com/pedrodornelas/Modelagem-Canal.

2 Definições de Variáveis Globais

As variáveis utilizadas estão representadas abaixo:

```
5   %% Simulation Params
6
7   % Enviroment - UMa-NLoS - number 4 sorted
8   env = 'UMa_NLoS';
9   % Frequency (GHz)
10  freq_ghz = 3;
11  c = 299792458; % m/s
12  lambda = c / (freq_ghz * 1e9);
13  % Multipath components
14  N = 100;
15  % Elevation Angle
16  phi_bar = pi/4;
17  % RX Velocity
18  v = 5; % m/s
```

Figura 1: Variáveis Globais

3 Espalhamento de Atraso (σ_τ)

Para gerar o termo de espalhamento de atraso do canal σ_τ e os termos de atraso τ_n foi criada uma função auxiliar que recebe o tipo de ambiente e retorna as estatísticas que serão utilizadas para gerar o espalhamento de atraso eficaz. Então, esta variável foi gerada utilizando a distribuição Gaussiana, em seguida, foi transformada em escala linear para ser utilizada nos demais pontos da simulação. Ainda, para gerar

os termos de atraso do canal, precisamos ainda do fator de proporcionalidade, que, a depender do ambiente, muda a intensidade do espalhamento de atraso. Dessa forma, com tais variáveis definidas, é possível gerar os termos de atraso.

```
%% Delay Spread RMS
% Stats
[delay_mean, delay_std] = get_delay_stats(freq_ghz, env);
% Gaussian - only one sample to represent
delay_spread_rms_log = normrnd(delay_mean, delay_std, [1,1]);
delay_spread = 10^(delay_spread_rms_log);
delay_spread_ns = 1e9*delay_spread

%% Generate Delay Samples
% Proportionality Factor
r_tau = get_prop_factor_delay(env);
mu_tau = delay_spread * r_tau;
% Delay samples
tau_n_absolute = exprnd(mu_tau, [N,1]);
tau_n_normalized = tau_n_absolute - min(tau_n_absolute);
tau_n = sort(tau_n_normalized);
% tau_n_ns = tau_n(1:5)*1e9
```

Figura 2: Espalhamento de Atraso e Amostras de Atraso

Para a definição do espalhamento de atraso geralmente se utiliza distribuições Gaussianas ou distribuições Log-Normais, pois os eventos da natureza, como o multipercurso, geralmente tem um grande fator de aleatoriedade, somando diversos eventos aleatórios. E segundo uma proposição, a soma de eventos aleatórios, ou seja, variáveis aleatórias, tendem para distribuições Gaussianas, e por consequência, a Log-Normal que a dual da distribuição Gaussiana.

A média do espalhamento de atraso diminui conforme o aumento da frequência pois quando aumentando a frequência, ou seja, aumentamos a banda, o sinal no tempo é estreitado, portanto, veremos que a média do espalhamento de atraso diminui. Também podemos atribuir que ao aumentar a frequência, fisicamente estamos tornando o canal menos suscetível à ter multipercurso, dado que há uma perda de percurso severa da potência conforme a distância aumenta, portanto, quando aumentamos a frequência, é provável de várias componentes multipercursos nem chegarem ao receptor, diminuindo a média para próximo das componentes que tem potências maiores e conseguem chegar ao receptor.

O espalhamento de atraso é uma medida do quanto o canal espalha o sinal no domínio de atraso, isto é, o quanto o canal gera componentes atrasadas do sinal transmitido. Assim, quanto maior o espalhamento de atraso, significa que as componentes multipercurso estão mais espalhadas no domínio de atraso, o que implica uma menor banda de coerência e isto significa que o canal filtra o sinal transmitido com mais severidade.

Por fim, as componentes de atraso τ_n são geradas utilizando uma distribuição de probabilidade exponencial. Esta distribuição tem maior probabilidade de ocorrência para valores menores, ou seja, teremos mais componentes multipercurso próximo do eixo y, e quando nos afastamos do eixo y a probabilidade de ocorrência é menor. Dessa forma, teremos mais componentes de atraso próximo ao eixo y da distribuição e menos componentes mais afastadas, ou seja, teremos mais componentes com um atraso menor e menos componentes com um atraso muito grande. Os valores irão depender da média da distribuição, no caso estamos utilizando $\mu_\tau = r_\tau \sigma_\tau$.

4 Potência Multipercurso

Para a geração dos termos de potência multipercurso foram gerados os termos de sombreamento, termos de potência e fator de rice. Após isto, pôde-se fazer a correspondência dos termos gerados com os termos de atraso, e então teremos o perfil de atraso de potência do canal. A geração dos termos e normalização da potência foram feitos da seguinte forma:

```
39  %% Generate Power Samples
40  % Power stats
41  [shadow_std, rice_mean, rice_std] = get_power_stats(env);
42  % Shadowing samples
43  shadow_samples = normrnd(0, shadow_std, [N,1]);
44  % Power samples
45  alpha_n_2 = exp(-tau_n .* (r_tau-1) ./ (r_tau*delay_spread)) .* 10.^(-shadow_samples ./ 10);
46  % alpha_n_2(1:5)
47
48  %% Rice factor
49  kr = 0;
50  kr_db = -inf;
51  if env == "UMi_LoS" || env == "UMa_LoS"
52      kr_db = normrnd(rice_mean, rice_std, [1,1]);
53      kr = 10 ^ (kr_db / 10);
54  end
55  kr
56
57  % Normalizing power
58  hat_omega_c = sum(alpha_n_2(2:N));
59  alpha_n_2 = (1/(kr + 1)) .* (alpha_n_2 ./ hat_omega_c);
60  alpha_n_2(1) = kr / (kr + 1);
61
62  % Verifying power = 1
63  omega_c = sum(alpha_n_2)
64  % Verifying Rice Factor
65  kr_check = alpha_n_2(1) ./ sum(alpha_n_2(2:N))
66  % Verifying Delay Spread
67  tau_bar = (1 / omega_c) * sum(tau_n .* alpha_n_2);
68  delay_spread
69  sigma_tau_check = sqrt((1 / omega_c) * sum(alpha_n_2 .* ((tau_n - tau_bar) .^ 2)))
```

Figura 3: Variáveis para Potência Multipercurso

O modelo do 3GPP 38.901 relaciona os termos de potência com os termos de atraso em uma relação de decaimento exponencial. Isto significa que as potências diminuem conforme os termos de atraso aumentam, ou seja, quanto maior o atraso da componente, menor será sua potência. Fisicamente sua explicação é lógica se pensarmos que quanto mais tempo a componente leva para chegar ao receptor, será dissipada mais potência durante o percurso.

O sombreamento é uma variação que pode ocorrer no sinal devido à objetos ou barreiras encontrados no percurso, ocasionando a perda de potência. Caso isto não ocorresse, a potência iria depender unicamente da distância percorrida pelo sinal, mas não é isto que vemos na prática, assim, essa variação ocasionada pelo sombreamento é atribuída aos sinais.

O fator de Rice caracteriza a relação entre a componente de visada direta e as componentes sem visada direta, relacionando-as com uma relação de potência. O fator de rice é gerado utilizando uma distribuição Gaussiana, com seu valor em dB, e em seguida, deve-se passar para a escala linear para realizar os cálculos e simulações.

O fator de Rice para meu cenário (UMa-NLoS) em linear seria 0, pois não temos visada direta. Porém, para verificar se o código estava fazendo uma correta normalização, verifiquei em um cenário UMa-LoS estes números. Na linha 55 tenho o fator de Rice gerado sendo mostrado e na linha 65 tenho a checagem do mesmo após as normalizações. Além disto, também achei interessante checar se o ganho do canal estava unitário após as normalizações, mostrado na linha 63. Isto gerou o seguinte:

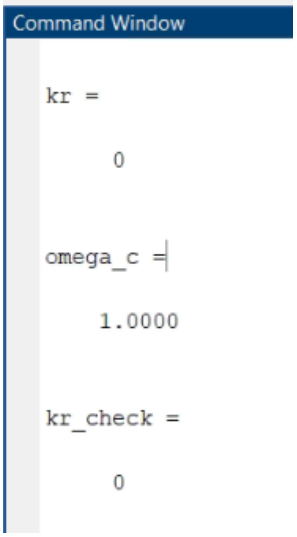


Figura 4: UMa-NLoS

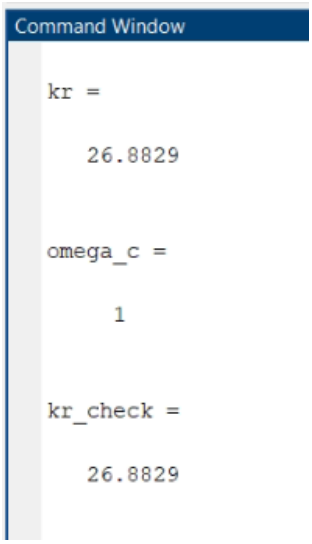


Figura 5: UMa-LoS

Após a geração e corretas conversões dos parâmetros acima citados, foi possível plotar a curva de perfil de de atraso de potência:

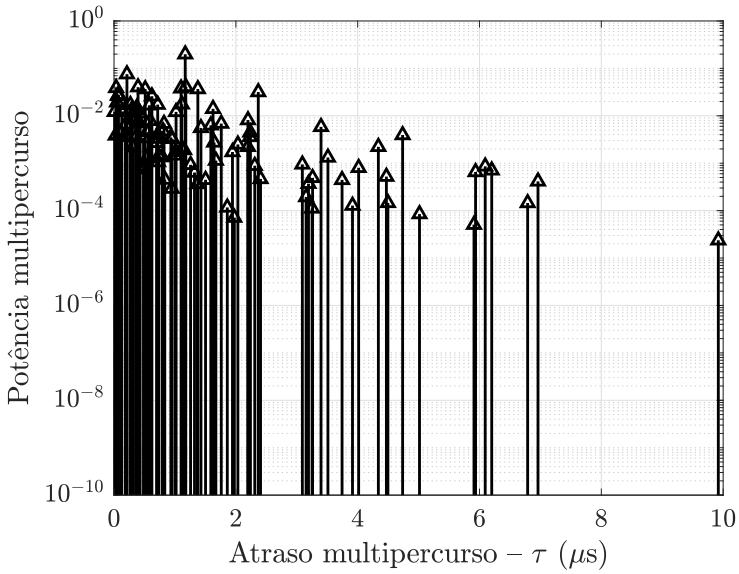


Figura 6: Perfil de Atraso de Potência

A potência parece bastante espalhada e não seguir o decaimento exponencial, mas se colocamos um gráfico utilizando os mesmos parâmetros com 1000 componentes, vemos o decaimento exponencial mais claramente:

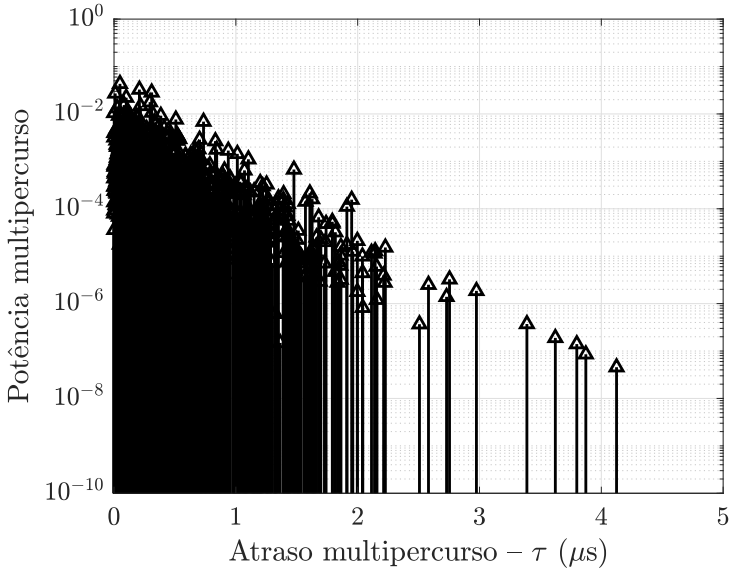


Figura 7: Perfil de Atraso de Potência - 1000 componentes

Também foi possível checar o espalhamento de atraso com os cálculos entre as linhas 66 a 69, o que resultou:

```
delay_spread =

   8.8316e-07


sigma_tau_check =

   8.3889e-07
```

Figura 8: Espalhamento de Atraso Gerado e Medido

Vale ressaltar que em várias simulações esse espalhamento ficou até pior, e isso decorre que para termos o valor gerado exatamente igual deveríamos ter infinitas componentes de atraso para convergir. Isto decorre do comportamento da lei dos grandes números.

5 Ângulos de chegada

Aqui trataremos de ângulos de chegada em azimute (plano horizontal) e em elevação (plano vertical) para compor as componentes angulares do sinal. Os ângulos

foram gerados da seguinte forma:

```
% -------------------------------------
%% Angles Stats
% -------------------------------------
[azimuth_mean, azimuth_std, elevation_mean, elevation_std] = get_angle_stats(freq_ghz, env);
% Azimuth Angles
sigma_theta_degree = 10 ^ (normrnd(azimuth_mean, azimuth_std, [1,1]));
sigma_theta_rad = sigma_theta_degree * (pi / 180);
max_alpha = max(alpha_n_2);
theta_n = 1.42 * sigma_theta_rad * sqrt(-log(alpha_n_2 ./ max_alpha));
% Adjusts
un = randsample([-1,1], N, true)';
% un(1:5)
yn = normrnd(0, sigma_theta_rad/7, [N, 1]);
% yn(1:5)
% Finally
theta_n = (un .* theta_n) + yn;
if env == "UMi_LoS" || env == "UMa_LoS"
    theta_n = theta_n - theta_n(1);
You, 2 hours ago | 1 author (You)
else
    theta_n(1) = 0;
end
```

Figura 9: Ângulos Azimutais

```
%% Elevation Angles
sigma_phi_degree = 10 ^ (normrnd(elevation_mean, elevation_std, [1,1]));
sigma_phi_rad = sigma_phi_degree * (pi / 180);
max_alpha = max(alpha_n_2);
phi_n = -sigma_phi_rad * log(alpha_n_2 ./ max_alpha);
% Adjusts
un = randsample([-1,1], N, true)';
% un(1:5)
yn = normrnd(0, sigma_phi_rad/7, [N, 1]);
% yn(1:5)
% Finally
phi_n = (un .* phi_n) + yn;
if env == "UMi_LoS" || env == "UMa_LoS"
    phi_n = phi_n - phi_n(1) + phi_bar;
You, 2 hours ago | 1 author (You)
else
    phi_n(1) = 0;
end
```

Figura 10: Ângulos em Elevação

Por fim, o significado físico dos ângulos de chegada irão definir como as componentes multipercurso irão incidir sobre o receptor. Estes ângulo vão alterar a fase do sinal recebido, o que irá prejudicar ou ajudar a recepção, respectivamente, no caso de interferências destrutivas e construtivas. Geralmente se utiliza uma distribuição Gaussiana justamente por depender de muitos efeitos aleatórios, dessa forma, do que se conhece até hoje, a distribuição Gaussiana seria a mais adequada.

Então, foi possível associar a potência das componentes com os ângulos:

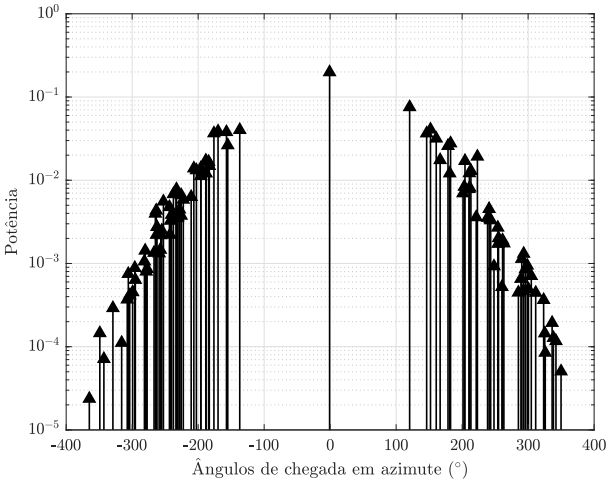


Figura 11: Espalhamento de Potência com Ângulos Azimutais

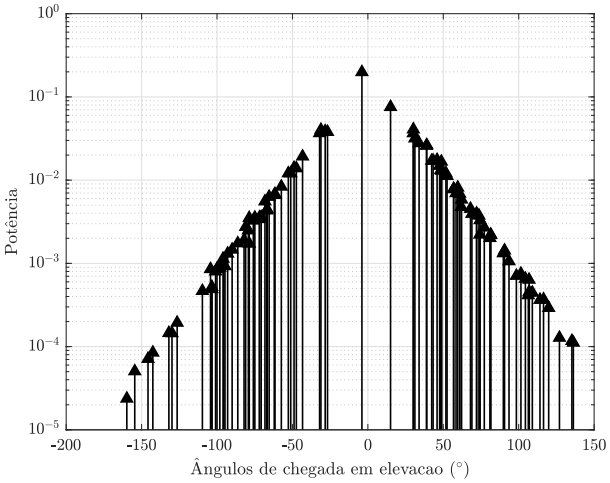


Figura 12: Espalhamento de Potência com Ângulos em Elevação

Com os ângulos gerados, foi possível então gerar os vetores de direção de chegada r_n:

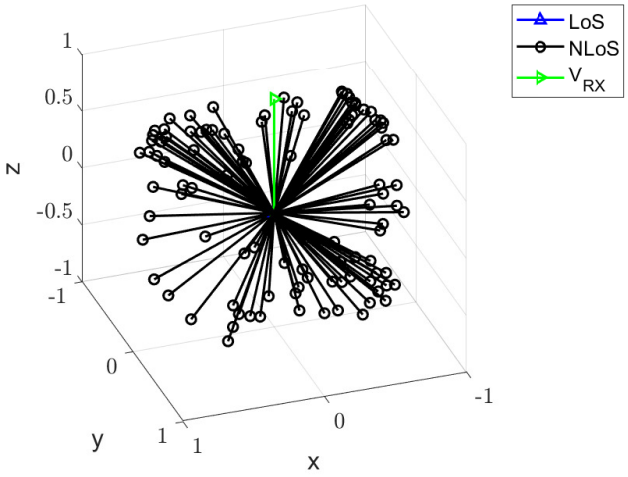


Figura 13: Vetores de Direção de Chegada

Observação: a próxima sessão referencia também o vetor direção de velocidade do receptor, foi plotado junto para facilitar a visualização. Como o cenário simulado foi UMa-NLoS, a componente de LoS não está representada pois o vetor dela precisa ser zerado. No caso de simular um ambiente com visada direta é possível ver o seu vetor direção representado.

6 Efeito Doppler

Para analisar o efeito doppler, ao invés de criar dois vetores velocidades, fixei as demais variáveis (espalhamento de atraso, fator de rice, ângulos de chegada) e rodei duas vezes a simulação, alterando a velocidade do receptor. Então, foi possível calcular os termos de desvio da seguinte forma:

```
259   % --------------------------------------
260 ~ %% Doppler Effects
261   % --------------------------------------
262   vn = (v / lambda) * (rn * vrx');
263   vn = sum(vn, 2);
```

Figura 14: Desvio Doppler

Note da figura acima que foi necessário calcular o produto interno. Isto foi feito em duas etapas, a primeira a multiplicação de um vetor linha (x, 1) por um vetor coluna (1, x), resultando em uma matriz (1, x), em seguida, a soma foi feita nas colunas, resultando no produto interno (1, 1).

Dessa forma, foi possível observar os termos dispersão de potência com desvio Doppler do canal para as duas velocidades:

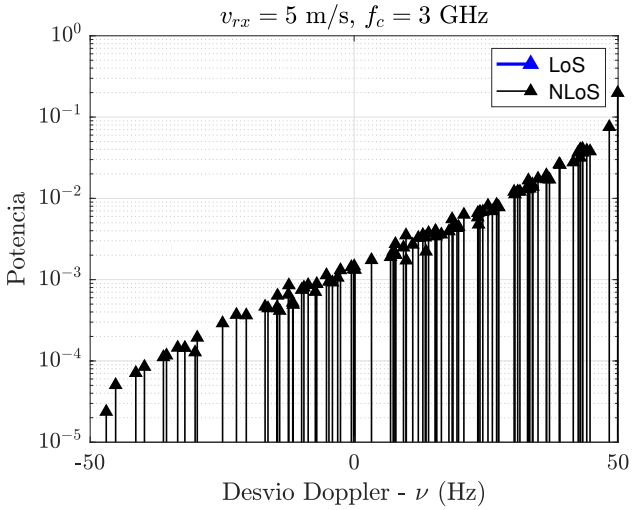


Figura 15: Dispersão de Potência Desvios Doppler $v = 5m/s$

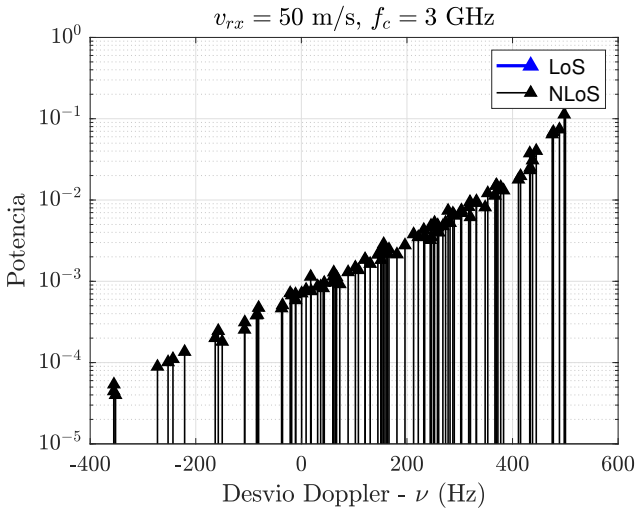
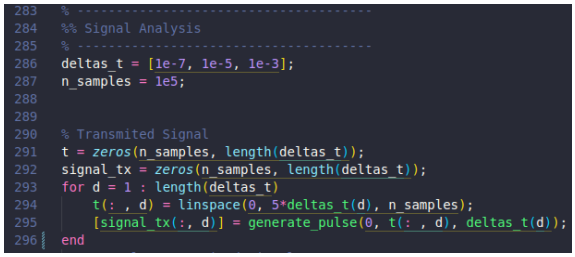


Figura 16: Dispersão de Potência Desvios Doppler $v = 50m/s$

Foi possível observar que os desvio doppler foram aproximadamente 10 vezes maiores, conforme o aumento também de 10 vezes da velocidades, seguindo o comportamento esperado.

7 Transmissão do Sinal

Foram gerados três pulsos retangulares de larga δt, e também foi necessário a geração de três eixos temporais para representar os pulsos, dado às diferenças de ordem de grandeza dos mesmos. Para isto, o código foi feito para iterar sobre os valores de δt e gerar o eixo temporal e o sinal transmitido por meio de uma função auxiliar que cria o pulso:

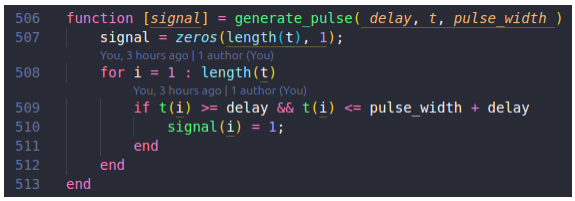


```
% ------------------------------------
%% Signal Analysis
% ------------------------------------
deltas_t = [1e-7, 1e-5, 1e-3];
n_samples = 1e5;


% Transmited Signal
t = zeros(n_samples, length(deltas_t));
signal_tx = zeros(n_samples, length(deltas_t));
for d = 1 : length(deltas_t)
    t(: , d) = linspace(0, 5*deltas_t(d), n_samples);
    [signal_tx(:, d)] = generate_pulse(0, t(: , d), deltas_t(d));
end
```

Figura 17: Geração dos Sinais Transmitidos

Função auxiliar para gerar o pulso:



```
function [signal] = generate_pulse( delay, t, pulse_width )
    signal = zeros(length(t), 1);
    You, 3 hours ago | 1 author (You)
    for i = 1 : length(t)
        You, 3 hours ago | 1 author (You)
        if t(i) >= delay && t(i) <= pulse_width + delay
            signal(i) = 1;
        end
    end
end
```

Figura 18: Função Auxiliar para Geração dos Pulsos

Em seguida, foi possível então determinar os três sinais recebidos, com base nos eixos temporais, nos termos de multipercurso, ângulos e todos os parâmetros anteriormente definidos. Na figura abaixo é possível ver como os sinais recebidos foram determinados:

Figura 19: Cálculo dos Sinais Recebidos

Note da figura acima que o sinal atraso foi gerado utilizando a função auxiliar do passo anterior, dado o atraso τ_n da componente atrasada. Para o cálculo da fase foi utilizado os desvios doppler, o eixo temporal e a frequência utilizada. Em seguida, o sinal espalhado foi gerado somando os termos atrasados, ou seja, somando o sinal recebido ao longo do eixo das componentes multipercurso, de forma que no tempo $t = 1$, teremos todas as potências das componentes multipercurso somadas neste intervalo. Em seguida é tirado o módulo para ter o sinal puramente real e então ele é normalizado para que possamos comparar com o sinal transmitido com potência unitária.

Assim, teremos os três sinais transmitidos e recebidos:

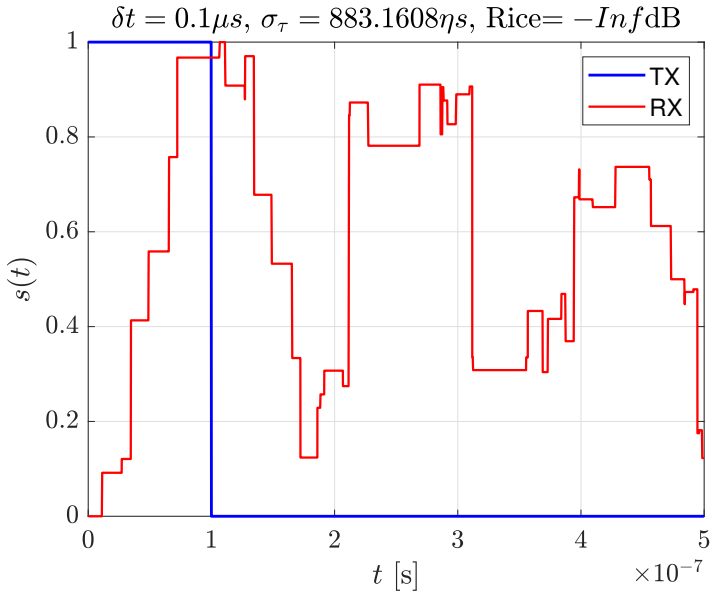


Figura 20: Sinal Recebido $\delta t = 10^{-7}s$

É possível perceber que o sinal com $\delta t = 10^{-7}s$ (20) é bastante filtrado pelo canal. Isto acontece pois a largura do pulso é bem pequena, na ordem do espalhamento de atraso, isto quer dizer que na frequência, este pulso tem uma largura de banda maior que a banda de coerência, portanto, o canal irá filtrar severamente o sinal recebido, portanto, conseguimos ver o sinal recebido bem deformado. Além do que, para o cenário UMa-NLoS, que não tem visada direta, as componentes multipercursos carregam toda a potência, portanto, ela se torna bem espalhada e a filtragem do canal se torna ainda mais impactante. Para este sinal, o canal é seletivo em frequência pois está espalhando bastante o sinal transmitido.

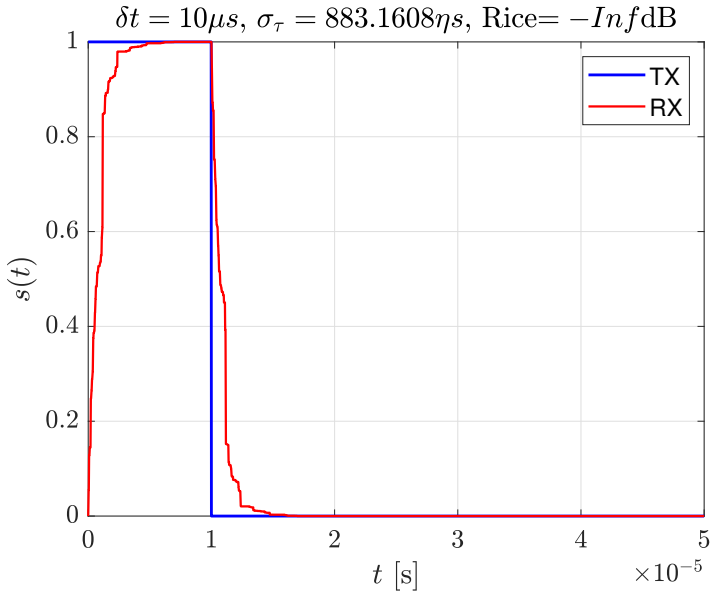


Figura 21: Sinal Recebido $\delta t = 10^{-5}s$

Já para o caso do sinal com $\delta t = 10^{-5}s$ (21) já não é tão filtrado pelo canal. Isto se dá pela relação entre espalhamento de atraso e largura do pulso, dado que a largura do pulso agora está a duas ordem maior que o espalhamento de atraso, na frequência o sinal transmitido foi estreitado, o que resulta em uma menor filtragem pela banda de coerência do canal. Mas ainda há uma filtragem, o que mostra que alguns componentes ainda não podem ser diferenciadas do ponto de vista do receptor, isto gera um certo ruído nas bordas (componentes de mais altas frequências).

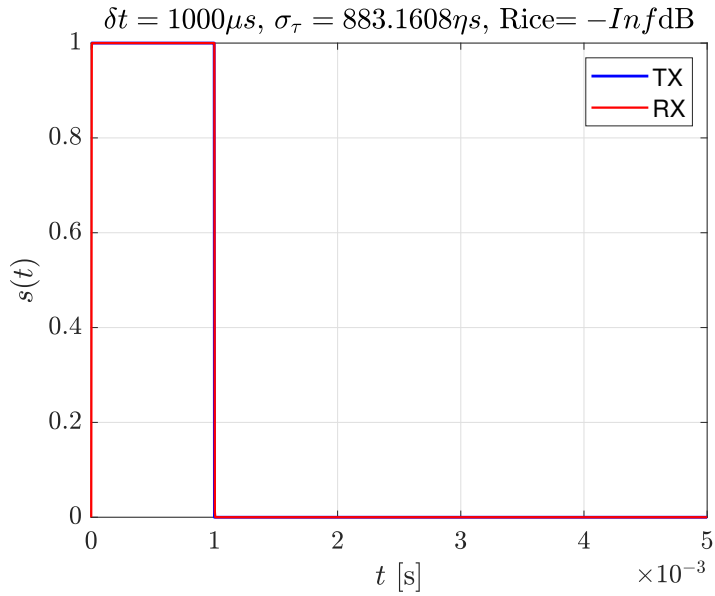


Figura 22: Sinal Recebido $\delta t = 10^{-3}s$

Para o caso com $\delta t = 10^{-3}s$ (22) é possível notar que o sinal quase não é filtrado pelo canal. Isto porque agora o espalhamento de atraso já é bem menor que a largura do pulso, isto resulta que a banda de coerência é bem maior que a banda do pulso transmitido, dado que ele foi alargado no tempo sua banda diminui, e agora quase, se não todas, as componentes multipercurso são identificadas pelo receptor sem que o sinal seja tão espalhado, tornando a recepção quase perfeita. Isto mostra um canal não seletivo em frequência, dado o sinal transmitido.

8 Tempo e Banda de Coerência do Canal

A função de autocorrelação do canal foi feita isolando frequência e tempo. Para isto, foram gerados os vetores da seguinte forma:

```
355  %% Coherency bandwidth and coherency time
356  n_samples = 1e4;
357  k = logspace(-3, 10, n_samples);
358  t = logspace(-6, 0, n_samples);
359
360  omega_c = sum( alpha_n_2 );
361  autocorr_freq_n = (1 / omega_c) .* alpha_n_2 .* exp( -2i * pi * tau_n .* k );
362  autocorr_freq = sum(autocorr_freq_n, 1);
363
364  autocorr_time_n = (1 / omega_c) .* alpha_n_2 .* exp( 2i * pi * vn .* t);
365  autocorr_time = sum(autocorr_time_n, 1);
```

Figura 23: Funções de Autocorrelações

Note que o eixo da frequência foi gerado utilizando k e o eixo de tempo foi representado por t. Então, para gerar a autocorrelação em frequência, zeramos os termos temporais. Já para a autocorrelação temporal, zera-se o termo de frequência.

Então, é possível encontrar a banda de coerência que mantém a taxa de correlação de 95% e 90%:

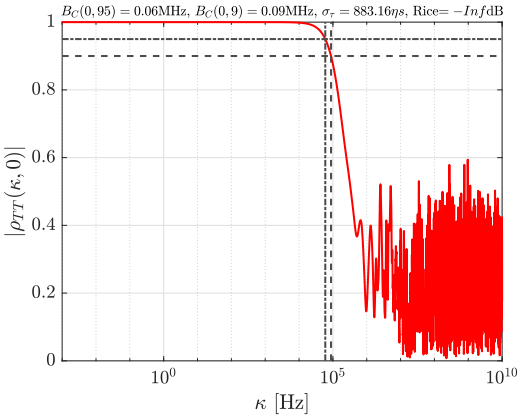


Figura 24: Banda de Coerência

Em seguida, também é possível encontrar o tempo de coerência que mantém a taxa de correlação de 95% e 90%, para as velocidades de $v = 5m/s$ e $v = 50m/s$:

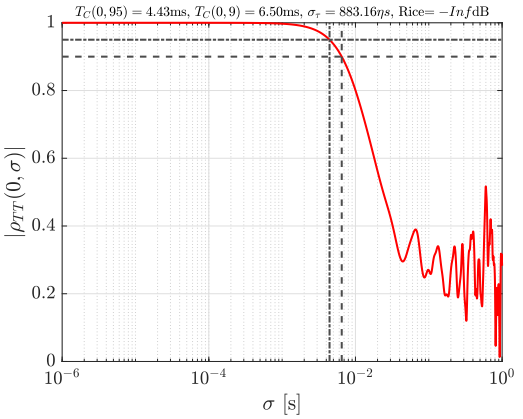


Figura 25: Tempo de Coerência $v = 5m/s$

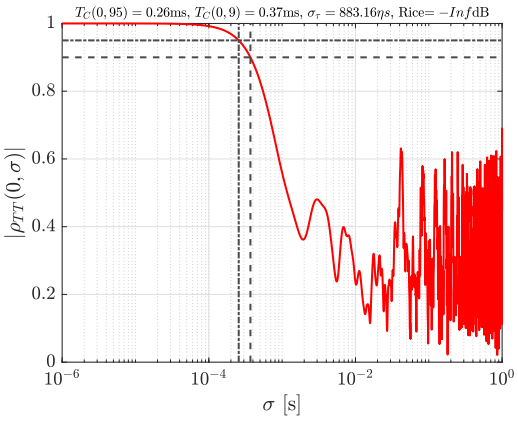


Figura 26: Tempo de Coerência $v = 50m/s$

Das figuras acima, note que o tempo de coerência para as duas porcentagens caiu na ordem de 10 vezes ao mudar a velocidade de $v = 5m/s$ para $v = 50m/s$, seguindo a proporção de aumento da velocidade. Como tive de simular duas vezes, mesmo mantendo os parâmetros de larga escala, os valores dos parâmetros das componentes mudaram (e devem mudar). Porém, caso tivesse utilizado exatamente o mesmo canal, veríamos uma diminuição exata de 10 vezes no tempo de coerência. Portanto, é se de notar que o efeito doppler pode ser bem nocivo ao canal de comunicação.