Part 1: Capacity gain in MIMO system

Table of Contents

Parameters

```
lambda = 0.06;  % ???
Nt_Nr = {[8, 8], [16, 16]};  % Cell array for pairs
dt = lambda / 2;
dr = lambda / 2;
L = 6;
SNR = -10:2:20;
num_MC = 1000;
x = randn(max(Nt_Nr{2}), 1) + 1j * randn(max(Nt_Nr{2}), 1);  % Random signal
noise_var = 10.^(-SNR / 10); % Noise variance calculation
qam = 4;
```

Plotting for (Nt, Nr) = (8, 8) and (16, 16)

```
for idx = 1:length(Nt_Nr)
    Nt = Nt_Nr{idx}(1);
    Nr = Nt_Nr{idx}(2);

    % Initialize storage for achievable rates (Monte Carlo averaging)
    rate_ZF_rich_avg = zeros(1, length(SNR));
    rate_LMMSE_rich_avg = zeros(1, length(SNR));
    rate_SVD_rich_avg = zeros(1, length(SNR));

    rate_ZF_sparse_avg = zeros(1, length(SNR));
    rate_LMMSE_sparse_avg = zeros(1, length(SNR));
    rate_SVD_sparse_avg = zeros(1, length(SNR));

    % Monte Carlo Trials
    for MC_id = 1:num_MC
        % Generate channels
        Hr = calculateRichChannel(Nr, Nt);  % Rich channel
        Hs = calculateSparseChannel(Nr, Nt, L, dt, dr, lambda);  % Sparse
channel

        % Generate symbols to transmit through transmit antennas
        input_bits = randi([0 1],Nt*log2(qam),1);
        x = qammod(input_bits, qam, 'InputType', 'bit', 'UnitAveragePower',
true);

        for snr_idx = 1:length(SNR)
```

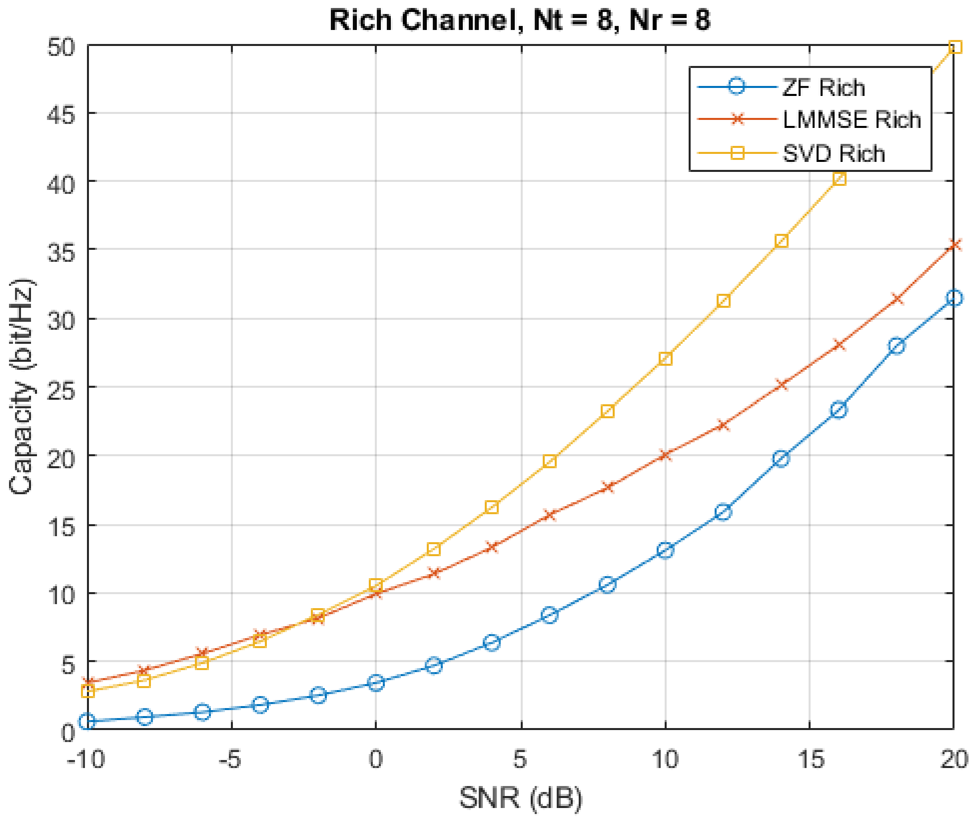
```
            snr_dB = SNR(snr_idx);
            SNR_linear = 10^(snr_dB / 10);  % Convert to linear scale
            noise_var = 1 / SNR_linear;  % Adjust noise variance based on SNR

            % ZF Combining
            [W_ZF_rich, capacity] = calculateZFCombiner(Hr, x, Nt, Nr,
SNR_linear);
            rate_ZF_rich_avg(snr_idx) = rate_ZF_rich_avg(snr_idx) + capacity;

            [W_ZF_sparse, capacity] = calculateZFCombiner(Hs, x, Nt, Nr,
SNR_linear);
            rate_ZF_sparse_avg(snr_idx) = rate_ZF_sparse_avg(snr_idx) +
capacity;

            % LMMSE Combining
            [W_LMMSE_rich, capacity] = calculateLMMSECombiner(Hr, x, Nt, Nr,
SNR_linear, noise_var);
            rate_LMMSE_rich_avg(snr_idx) = rate_LMMSE_rich_avg(snr_idx) +
capacity;

            [W_LMMSE_sparse, capacity] = calculateLMMSECombiner(Hs, x, Nt,
Nr, SNR_linear, noise_var);
            rate_LMMSE_sparse_avg(snr_idx) = rate_LMMSE_sparse_avg(snr_idx)
+ capacity;

            % SVD Combining
            [W_SVD_rich, capacity] = calculateSVDCombiner(Hr, x, Nt, Nr,
SNR_linear);
            rate_SVD_rich_avg(snr_idx) = rate_SVD_rich_avg(snr_idx) +
capacity;

            [W_SVD_sparse, capacity] = calculateSVDCombiner(Hs, x, Nt, Nr,
SNR_linear);
            rate_SVD_sparse_avg(snr_idx) = rate_SVD_sparse_avg(snr_idx) +
capacity;
        end
    end

    % Average over the Monte Carlo trials
    rate_ZF_rich_avg = rate_ZF_rich_avg / num_MC;
    rate_LMMSE_rich_avg = rate_LMMSE_rich_avg / num_MC;
    rate_SVD_rich_avg = rate_SVD_rich_avg / num_MC;

    rate_ZF_sparse_avg = rate_ZF_sparse_avg / num_MC;
    rate_LMMSE_sparse_avg = rate_LMMSE_sparse_avg / num_MC;
    rate_SVD_sparse_avg = rate_SVD_sparse_avg / num_MC;

    % Plotting results for this (Nt, Nr) pair
    figure;
    plot(SNR, rate_ZF_rich_avg, '-o', 'DisplayName', 'ZF Rich');
    hold on;
    plot(SNR, rate_LMMSE_rich_avg, '-x', 'DisplayName', 'LMMSE Rich');
    plot(SNR, rate_SVD_rich_avg, '-s', 'DisplayName', 'SVD Rich');
    title(['Rich Channel, Nt = ', num2str(Nt), ', Nr = ', num2str(Nr)]);
```

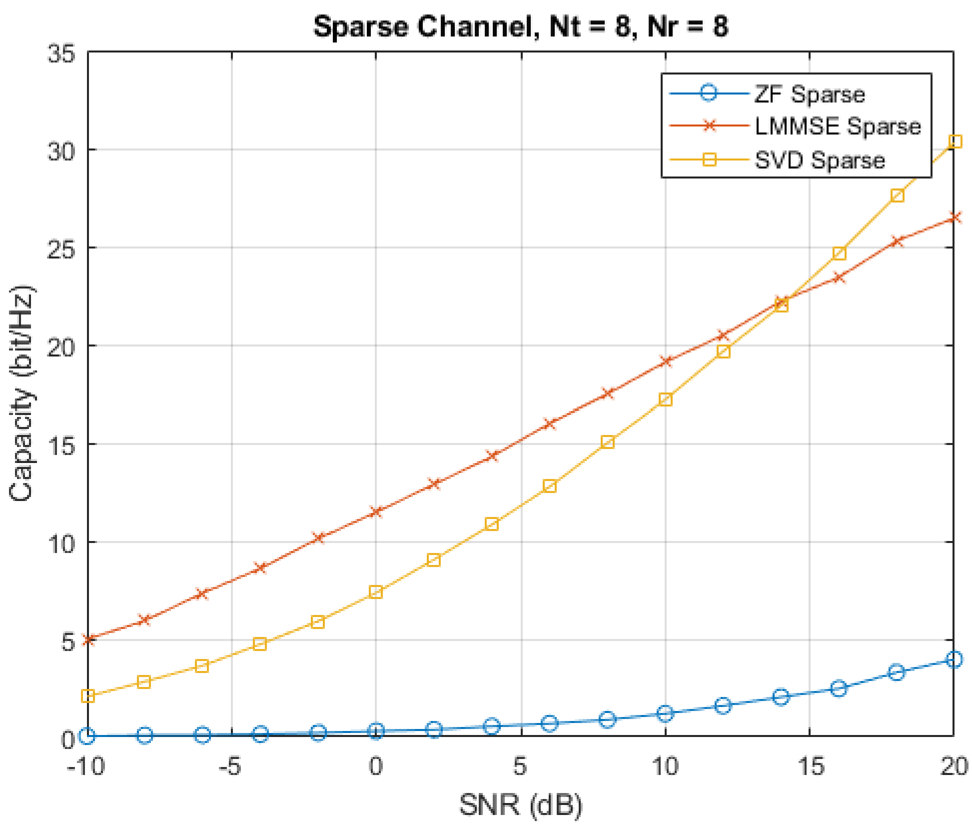
```
    xlabel('SNR (dB)');
    ylabel('Capacity (bit/Hz)');
    legend;
    grid on;

    figure;
    plot(SNR, rate_ZF_sparse_avg, '-o', 'DisplayName', 'ZF Sparse');
    hold on;
    plot(SNR, rate_LMMSE_sparse_avg, '-x', 'DisplayName', 'LMMSE Sparse');
    plot(SNR, rate_SVD_sparse_avg, '-s', 'DisplayName', 'SVD Sparse');
    title(['Sparse Channel, Nt = ', num2str(Nt), ', Nr = ', num2str(Nr)]);
    xlabel('SNR (dB)');
    ylabel('Capacity (bit/Hz)');
    legend;
    grid on;
end
```

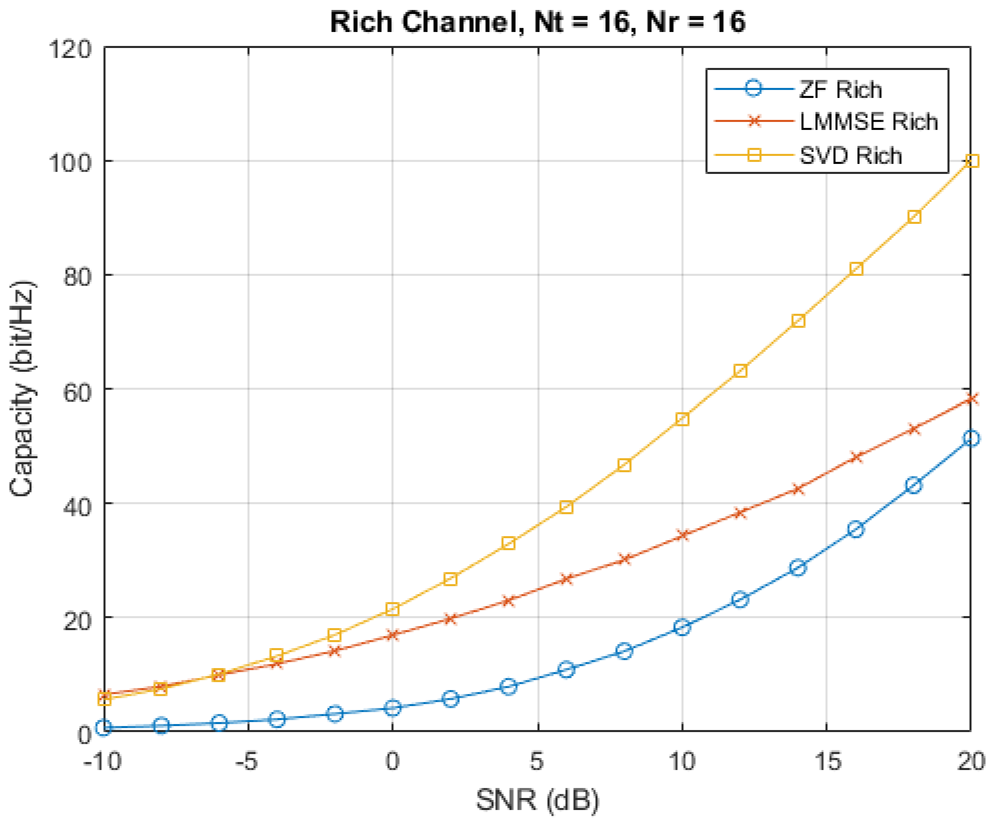
Functions

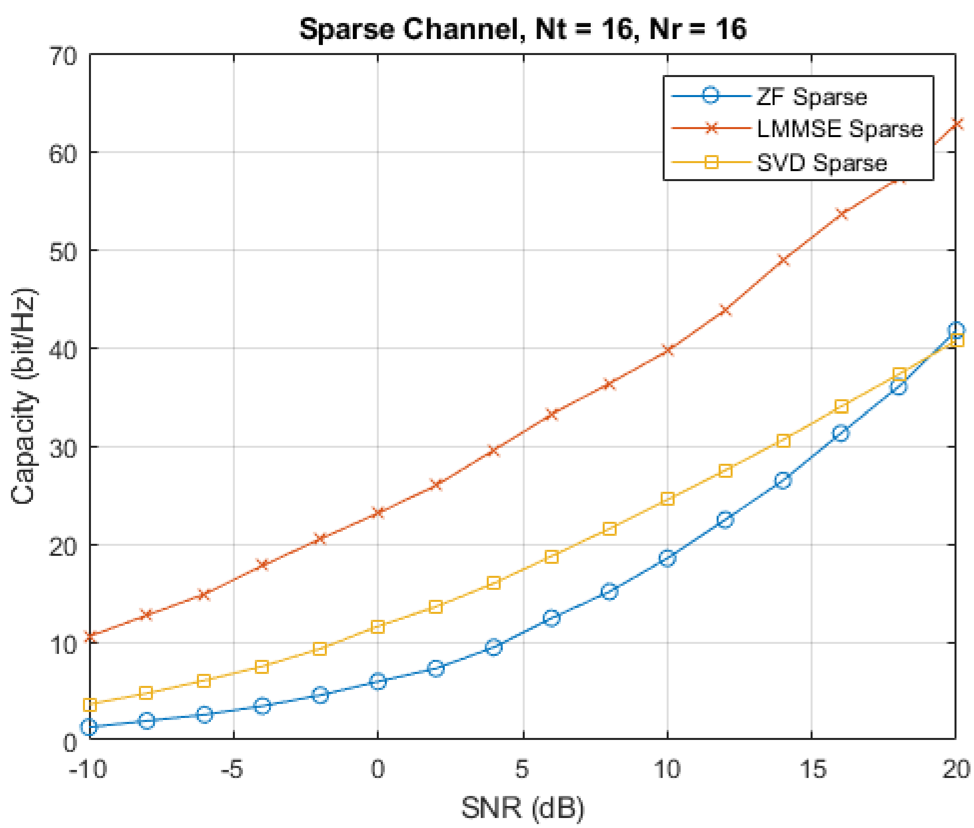
```
function Hr = calculateRichChannel(Nr, Nt)
    % Generate real and imaginary parts independently from N(0, 1/2)
    real_part = sqrt(1/2) * randn(Nr, Nt);
    imag_part = sqrt(1/2) * randn(Nr, Nt);

    % Combine to form complex Gaussian elements
    Hr = real_part + 1j * imag_part;
end

function Hs = calculateSparseChannel(Nr, Nt, L, dt, dr, lambda)
    Hs = zeros(Nr, Nt);  % Initialize the sparse channel matrix

    scale_factor = sqrt(Nt * Nr / L);

    for i = 1:L
        % Generate path gain (complex Gaussian random variable)
        alpha_i = sqrt(1/2) * (randn() + 1i*randn());   % N(0, 1) complex
normal

        % Generate AoA (theta_i) and AoD (phi_i) uniformly in [-pi/2, pi/2]
        theta_i = (-pi/2) + (pi) * rand();
        phi_i = (-pi/2) + (pi) * rand();

        % Calculate aRx(theta_i) spatial response vector
        aRx = sqrt(1/Nr) * exp(-1i * pi * (0:(Nr-1))' * sin(theta_i));

        % Calculate aTx(phi_i) spatial response vector
        aTx = sqrt(1/Nt) * exp(-1i * pi * (0:(Nt-1))' * sin(phi_i));

        % Update the sparse channel matrix
        Hs = Hs + alpha_i * (aRx * aTx');
    end

    Hs = scale_factor * Hs;
```

```
end

function capacity = calculateCapacity(x_hat, n_hat)
    x_hat = abs(x_hat).^2;
    n_hat = abs(n_hat).^2;
    capacity = sum(log2((x_hat./n_hat) + 1));
end

function [W_ZF, capacity] = calculateZFCombiner(H, x, Nt, Nr, SNR_linear)
    W_ZF = pinv(H);

    % Generate complex noise with noise power corresponding to SNR
    noise_power = (norm(H*x, 'fro')^2)/SNR_linear;
    noise = sqrt(1/2) * (randn(Nr,1) + 1j*randn(Nr,1));
    noise = (noise/norm(noise, 'fro')) * sqrt(noise_power);

    capacity = calculateCapacity(W_ZF*H*x, W_ZF*noise);
end


function [W_LMMSE, capacity] = calculateLMMSECombiner(H, x, Nt, Nr,
SNR_linear, noise_var)
    % Calculate the LMMSE combining matrix
    W_LMMSE = inv(H'*H+(1/SNR_linear)*eye(Nt))*H';

    % Generate complex noise with noise power corresponding to SNR
    noise_power = (norm(H*x, 'fro')^2)/SNR_linear;
    noise = sqrt(1/2) * (randn(Nr,1) + 1j*randn(Nr,1));
    noise = (noise/norm(noise, 'fro')) * sqrt(noise_power);

    capacity = calculateCapacity(W_LMMSE*H*x, W_LMMSE*noise);
end


function [W_SVD, capacity] = calculateSVDCombiner(H, x, Nt, Nr, SNR_linear)
    % Perform SVD on the channel matrix H
    [U, S, V] = svd(H); % H = UEV^H

    % W_SVD = U^H
    W_SVD = U';

    noise_power = (norm(H*V*x, 'fro')^2)/SNR_linear;
    noise = sqrt(noise_power) * sqrt(1/2) * (randn(Nr,1) + 1j*randn(Nr,1));
    noise = (noise/norm(noise, 'fro')) * sqrt(noise_power);

    capacity = calculateCapacity(W_SVD*H*V*x, W_SVD*noise);
end
```


Rich Channel, Nt = 8, Nr = 8
ZF Rich
LMMSE Rich
SVD Rich
Capacity (bit/Hz)
SNR (dB)
50
45
40
35
30
25
20
15
10
5
0
-10
-5
0
5
10
15
20


Sparse Channel, Nt = 8, Nr = 8
ZF Sparse
LMMSE Sparse
SVD Sparse
Capacity (bit/Hz)
SNR (dB)
35
30
25
20
15
10
5
0
-10
-5
0
5
10
15
20


Rich Channel, Nt = 16, Nr = 16
ZF Rich
LMMSE Rich
SVD Rich
Capacity (bit/Hz)
SNR (dB)
0
20
40
60
80
100
120
-10
-5
0
5
10
15
20


Sparse Channel, Nt = 16, Nr = 16
ZF Sparse
LMMSE Sparse
SVD Sparse
Capacity (bit/Hz)
SNR (dB)
0
10
20
30
40
50
60
70
-10
-5
0
5
10
15
20

Published with MATLAB® R2024a