
Part 2: Interference mitigation techniques for antenna array

Table of Contents

Parameters	1
Setup	1
(a) MRC combiner	2
Plot of the SINR vs σ^2	2
(b) MVC combiner	3
Plot of the SINR vs σ^2	4
MRC and MVC Combiners	4
Explanation	4
(c) MVC with imperfect channel estimation	5
Plot of the SINR vs σ^2	6
MRC, MVC, and Imperfect MVC Combiners	6
Explanation	6
(d) A variant of the minimum variance combiner: estimation of h_2	8
Compare estimated ϕ_2 with true ϕ_2	9
Explanation	9
(e) A variant of the minimum variance combiner: low-rank approximation	14
Plot of the SINR vs σ^2	14
Imperfect MVC and Low-Rank MVC Combiners	14
All combiners	15
Explanation	15
Functions	17

Parameters

```
N = 16; % Number of antennas
phi1 = 30 * pi / 180; % Angle for UE-1 in radians
phi2 = 40 * pi / 180; % Angle for UE-2 in radians.
true_phi2 = 40;
delta_phi1 = 1 * pi / 180; % Angle for UE-2 in radians
phi_range = -90:0.2:90;
T = 1000; % Number of symbols
P_s1 = 1; % Power of signal s1(t)
P_s2 = 10; % Magnitude of signal s2(t)
sigma2 = 1e-3; % Noise power
num_MC = 1000;
num_MC_faster = 10;
SNR_dB = -20:10:40; % SNR range in dB
zeta = 0.0001;
```

Setup

```
noise_power_dB = -SNR_dB; % Noise Power dB = -SNR dB
SNR = 10.^(SNR_dB/10); % SNR_dB --> SNR linear scale
```

```
% Channel vectors for UE-1 and UE-2
h1 = calculate_ULA_response(N, phi1); % N x 1 channel vector for UE-1
h2 = calculate_ULA_response(N, phi2); % N x 1 channel vector for UE-2

% Generate 4-QAM symbols for s1(t) and s2(t)
s1 = calculate_QAM_symbols(T, P_s1); % 4-QAM symbols for UE-1
s2 = calculate_QAM_symbols(T, P_s2); % 4-QAM symbols for UE-2
```

(a) MRC combiner

```
% Combiner
mrc = generate_MRC_combiner(h1);

% Calculate SINR over SNR
SINR_dB_mrc = zeros(size(SNR_dB)); % Preallocate SINR in dB

% Number of noise power values
num_vals = length(noise_power_dB);
for idx = 1:num_vals
    % Calculate noise power from SNR
    sigma2 = 1 / SNR(idx);

    % Initialize accumulators for SINR across trials
    SINR_accum = 0;

    for trial = 1:num_MC
        % Generate noise matrix for each trial
        n = generate_noise(N, T, sigma2);

        % Generate received signal
        x = calculate_received_signal_array(h1, h2, s1, s2, N, T, n);

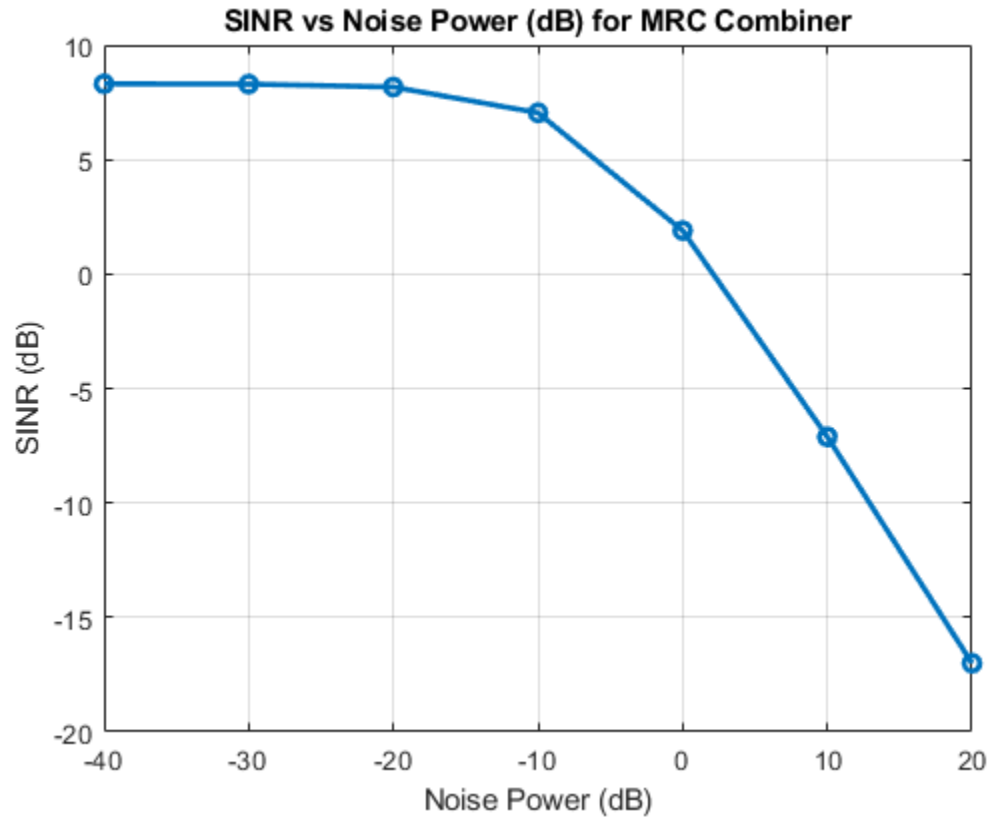
        % Compute SINR for the MRC combiner
        SINR_trial_dB = calculate_SINR(mrc, x, h1, h2, s1, s2, n, T);

        % Accumulate SINR results over trials
        SINR_accum = SINR_accum + 10^(SINR_trial_dB / 10); % Convert to
linear scale for averaging
    end

    % Average SINR over trials and convert back to dB
    mean_SINR = SINR_accum / num_MC;
    SINR_dB_mrc(idx) = 10 * log10(mean_SINR); % Convert back to dB
end
```

Plot of the SINR vs σ^2

```
plot_SINR_over_noise_power(SINR_dB_mrc, noise_power_dB, "MRC Combiner");
```



(b) MVC combiner

```
% Calculate SINR over SNR
SINR_db_mvc = zeros(size(SNR_db)); % Preallocate SINR in dB

% Number of noise power values
num_vals = length(noise_power_db);
for idx = 1:num_vals
    % Calculate noise power from SNR
    sigma2 = 1 / SNR(idx);

    % Initialize accumulators for SINR across trials
    SINR_accum = 0;

    for trial = 1:num_MC
        % Generate noise matrix for each trial
        n = generate_noise(N, T, sigma2);

        % Generate received signal
        x = calculate_received_signal_array(h1, h2, s1, s2, N, T, n);

        % Compute SINR for the MVC combiner
        mvc = generate_MVC_combiner(x, h1, N, T);
        SINR_trial_db = calculate_SINR(mvc, x, h1, h2, s1, s2, n, T);
```

```
% Accumulate SINR results over trials
SINR_accum = SINR_accum + 10^(SINR_trial_dB / 10); % Convert to
linear scale for averaging
end

% Average SINR over trials and convert back to dB
mean_SINR = SINR_accum / num_MC;
SINR_dB_mvc(idx) = 10 * log10(mean_SINR); % Convert back to dB
end
```

Plot of the SINR vs σ^2

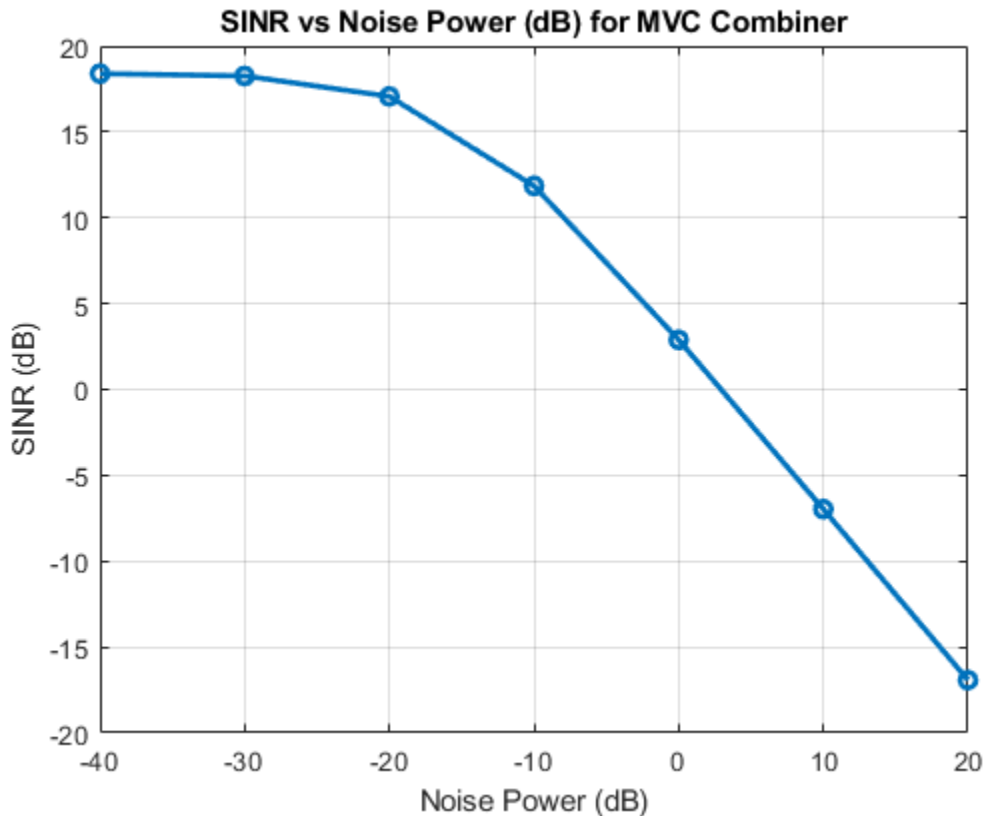
```
plot_SINR_over_noise_power(SINR_dB_mvc, noise_power_dB, "MVC Combiner");
```

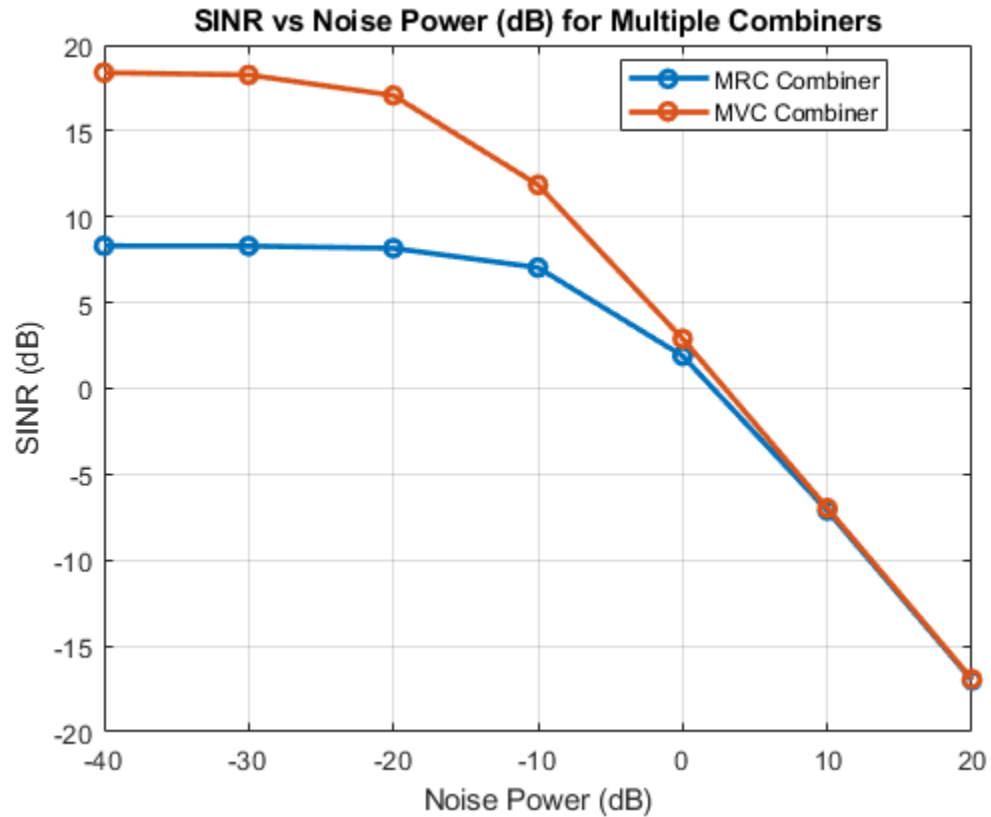
MRC and MVC Combiners

```
SINR_dB_list = {SINR_dB_mrc, SINR_dB_mvc};
combiner_names = {"MRC Combiner", "MVC Combiner"};
plot_SINR_over_noise_power_compare(SINR_dB_list, noise_power_dB,
combiner_names)
```

Explanation

As you can see between the MRC and MVC, initially, the MVC has a higher SINR than the MRC. However, at approximately Noise Power = 0 dB, they merge into similar SINR values. Both trend similarly.





(c) MVC with imperfect channel estimation

```
% Calculate SINR over SNR
SINR_dB_imperfect_mvc = zeros(size(SNR_dB)); % Preallocate SINR in dB

% Number of noise power values
num_vals = length(noise_power_dB);
for idx = 1:num_vals
    % Calculate noise power from SNR
    sigma2 = 1 / SNR(idx);

    % Initialize accumulators for SINR across trials
    SINR_accum = 0;

    for trial = 1:num_MC
        % Generate noise matrix for each trial
        n = generate_noise(N, T, sigma2);

        % Generate received signal
        x = calculate_received_signal_array(h1, h2, s1, s2, N, T, n);

        % Compute SINR for the MVC combiner
        imperfect_mvc = generate_imperfect_MVC(x, N, T, delta_phi1, phi1);
        SINR_trial_dB = calculate_SINR(imperfect_mvc, x, h1, h2, s1, s2, n,
T);
```

```
% Accumulate SINR results over trials
SINR_accum = SINR_accum + 10^(SINR_trial_dB / 10); % Convert to
% linear scale for averaging
end

% Average SINR over trials and convert back to dB
mean_SINR = SINR_accum / num_MC;
SINR_dB_imperfect_mvc(idx) = 10 * log10(mean_SINR); % Convert back to dB
end
```

Plot of the SINR vs σ^2

```
plot_SINR_over_noise_power(SINR_dB_imperfect_mvc, noise_power_dB, "Imperfect
MVC Combiner");
```

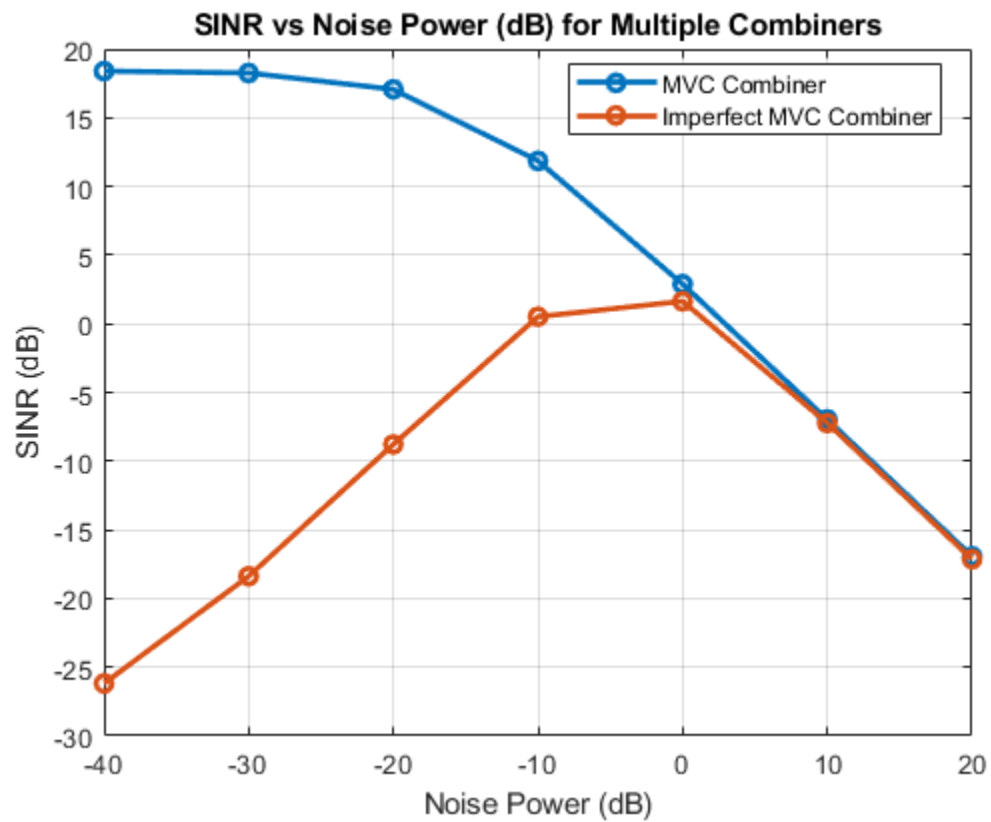
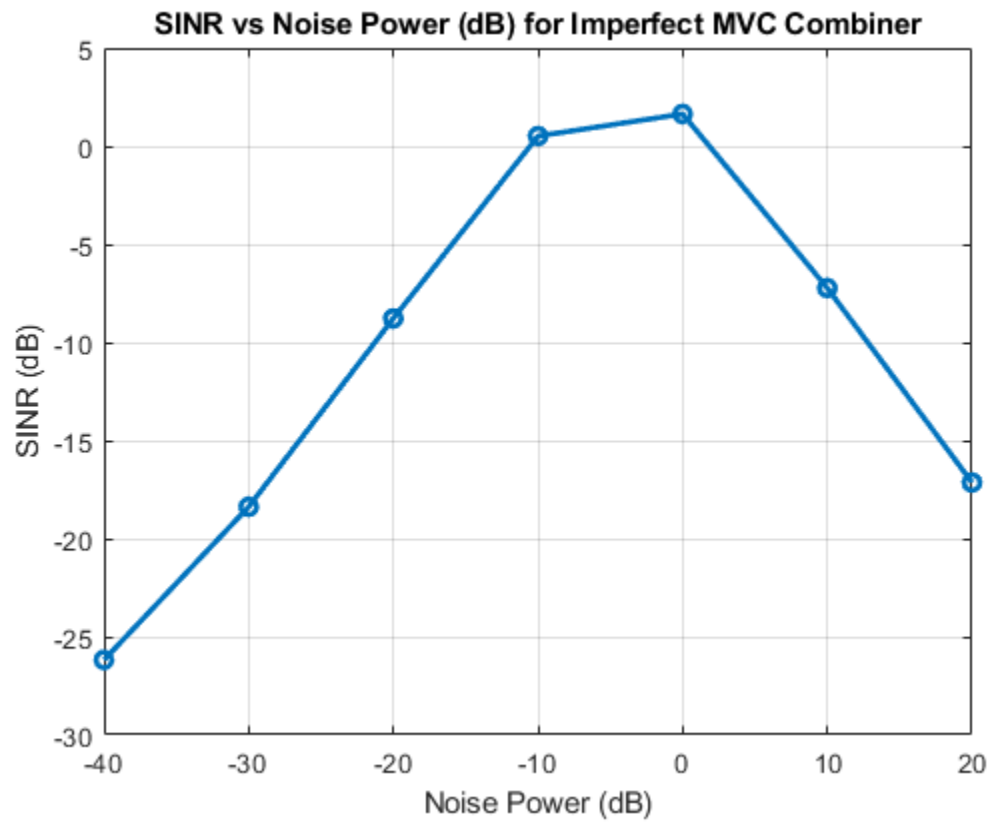
MRC, MVC, and Imperfect MVC Combiners

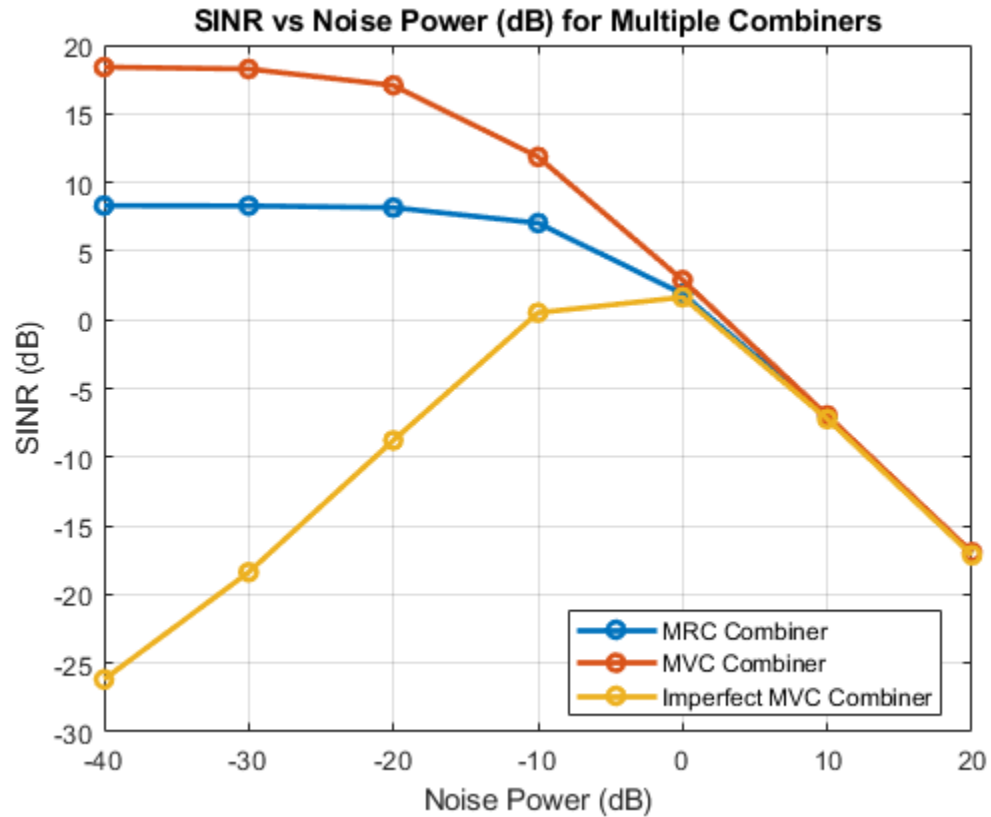
```
SINR_dB_list = {SINR_dB_mvc, SINR_dB_imperfect_mvc};
combiner_names = {"MVC Combiner", "Imperfect MVC Combiner"};
plot_SINR_over_noise_power_compare(SINR_dB_list, noise_power_dB,
combiner_names)

SINR_dB_list = {SINR_dB_mrc, SINR_dB_mvc, SINR_dB_imperfect_mvc};
combiner_names = {"MRC Combiner", "MVC Combiner", "Imperfect MVC Combiner"};
plot_SINR_over_noise_power_compare(SINR_dB_list, noise_power_dB,
combiner_names)
```

Explanation

In comparison to the MVC and MRC which decreases as Noise Power (dB) increases, the Imperfect MVC Combiner starts out with a low SINR value and increases until it merges with the MRC and MVC combiners at Noise Power = 0 dB and then all three decreases.





(d) A variant of the minimum variance combiner: estimation of h_2

```
num_vals = length(noise_power_dB);
estimates = zeros(num_vals, 1);
for idx = 1:num_vals
    sigma2 = 1/SNR(idx);

    estimated_phi2 = 0;
    for trial = 1:num_MC_faster
        % Generate noise matrix for each trial
        n = generate_noise(N, T, sigma2);

        % Generate received signal
        x = calculate_received_signal_array(h1, h2, s1, s2, N, T, n);

        % get estimated phi2 for each trial
        [w_phi, r_phi, phi2_hat] = estimate_phi2(x, N, T, phi_range);
        estimated_phi2 = estimated_phi2 + phi2_hat;
    end
    estimated_phi2 = estimated_phi2 / num_MC_faster;
    estimates(idx) = estimated_phi2;

    % Plot E|w(phi)'x(t)|^2 vs phi, for each sigma
```



```
figure;
plot(phi_range, r_phi, 'LineWidth', 2);
xlabel('phi (degrees)');
ylabel('E|w(\phi)' 'x(t)|^2');
title(['Plot for noise power = ', num2str(noise_power_dB(idx)), ' dB']);
grid on;
end
```

Compare estimated phi2 with true phi2

```
estimates
errors = abs(estimates - true_phi2); % Calculate the absolute error

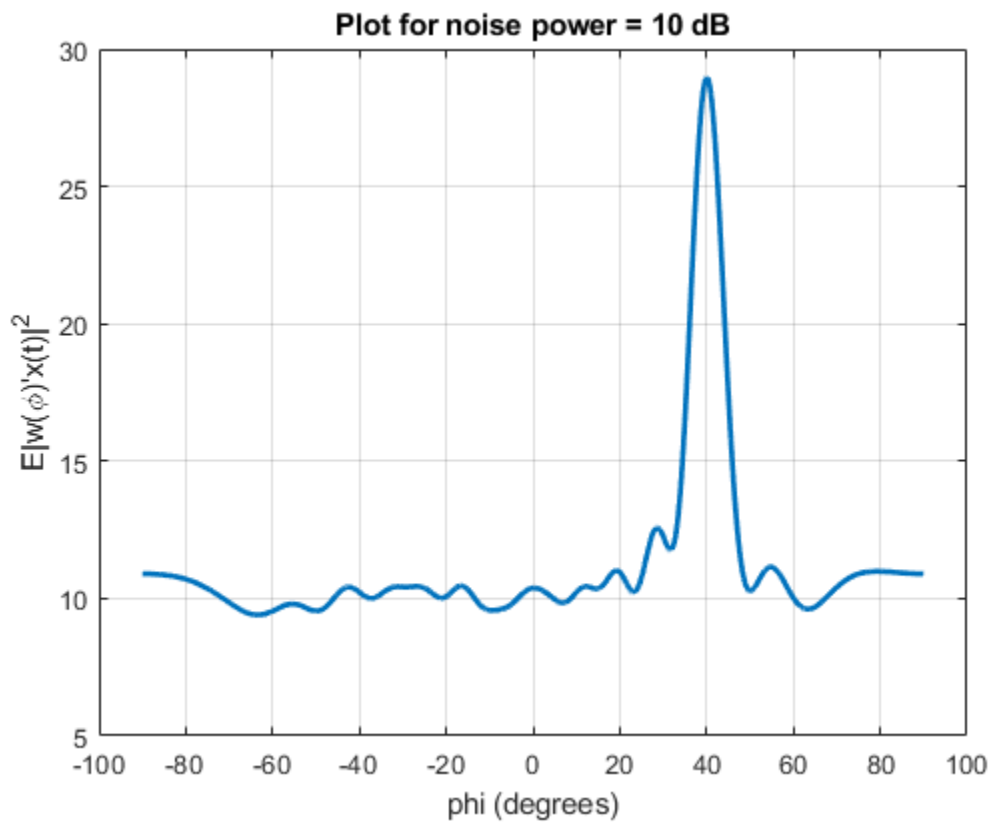
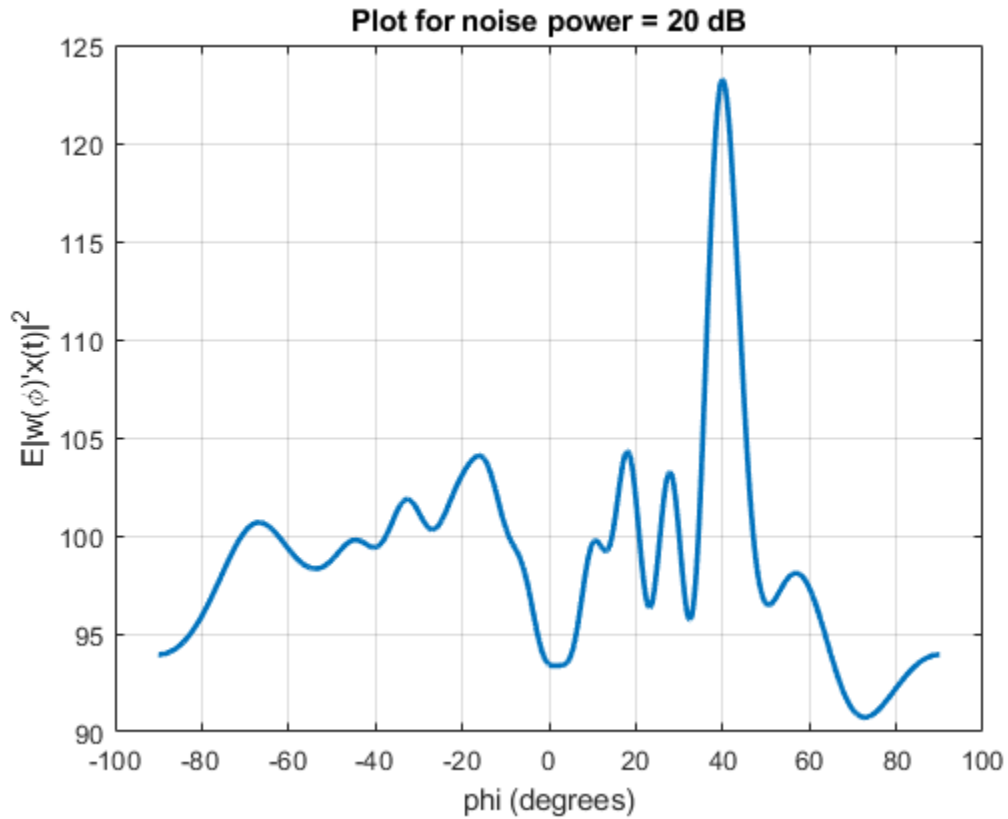
figure;
plot(noise_power_dB, errors, '-o', 'LineWidth', 2);
xlabel('Noise power (dB)');
ylabel('Absolute Error (degrees)');
title('Error between Estimated \phi_2 and True \phi_2 = 40^\circ');
grid on;
```

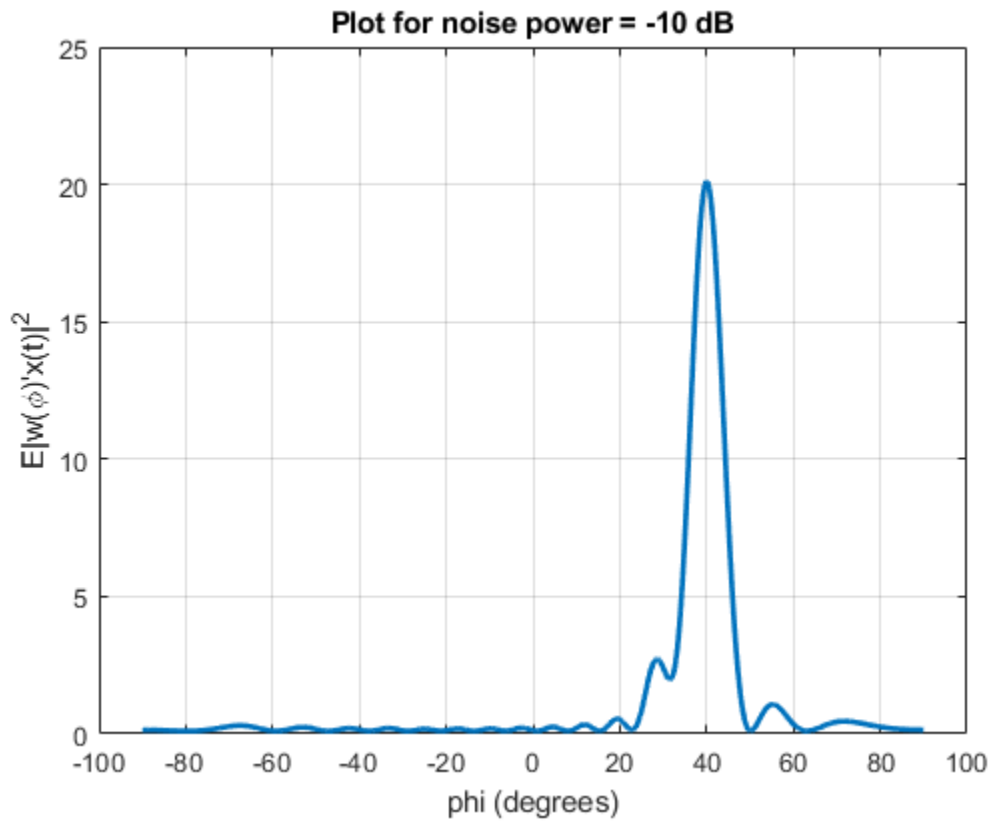
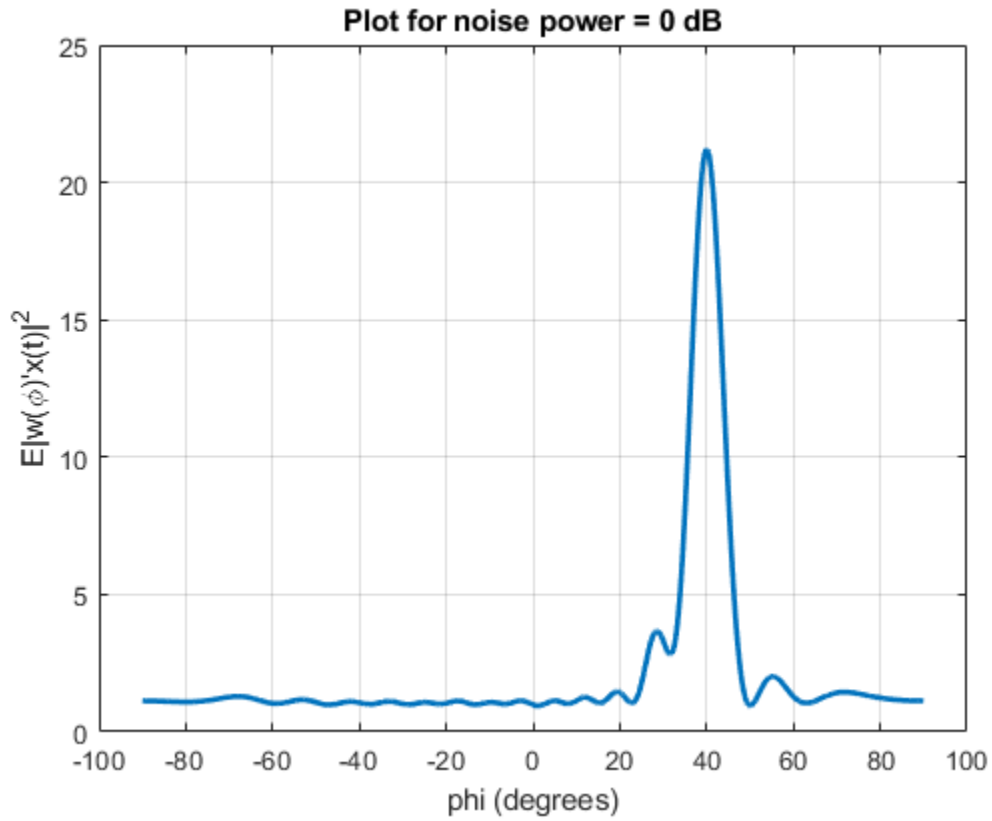
Explanation

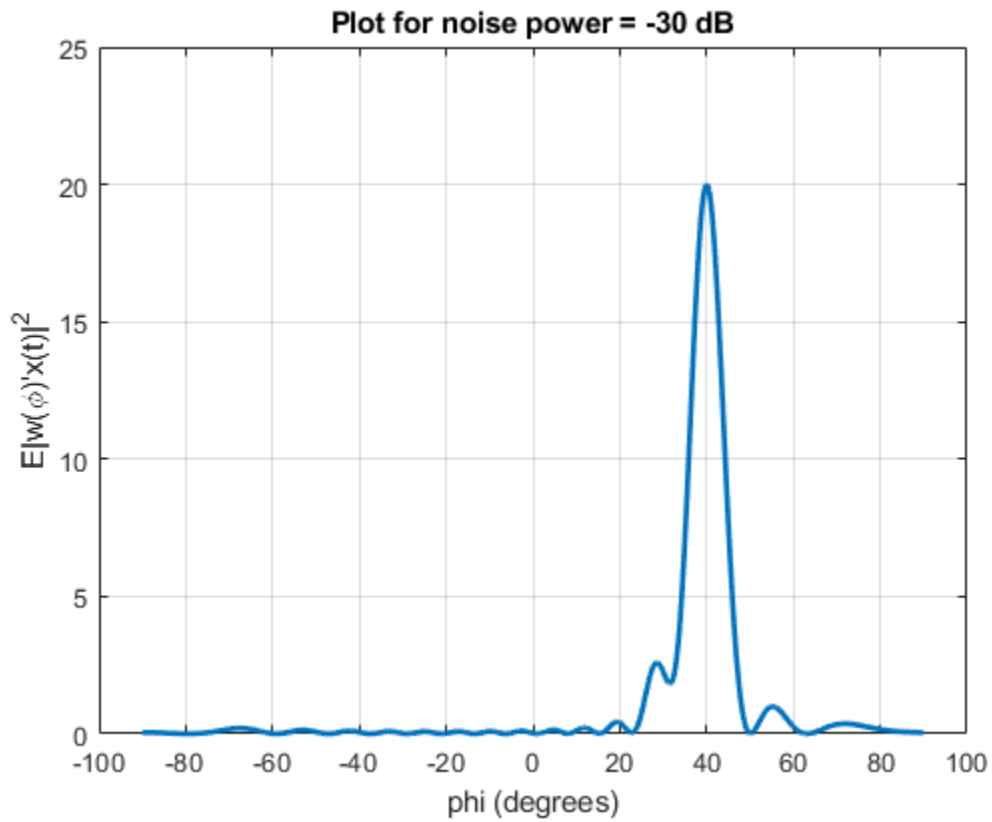
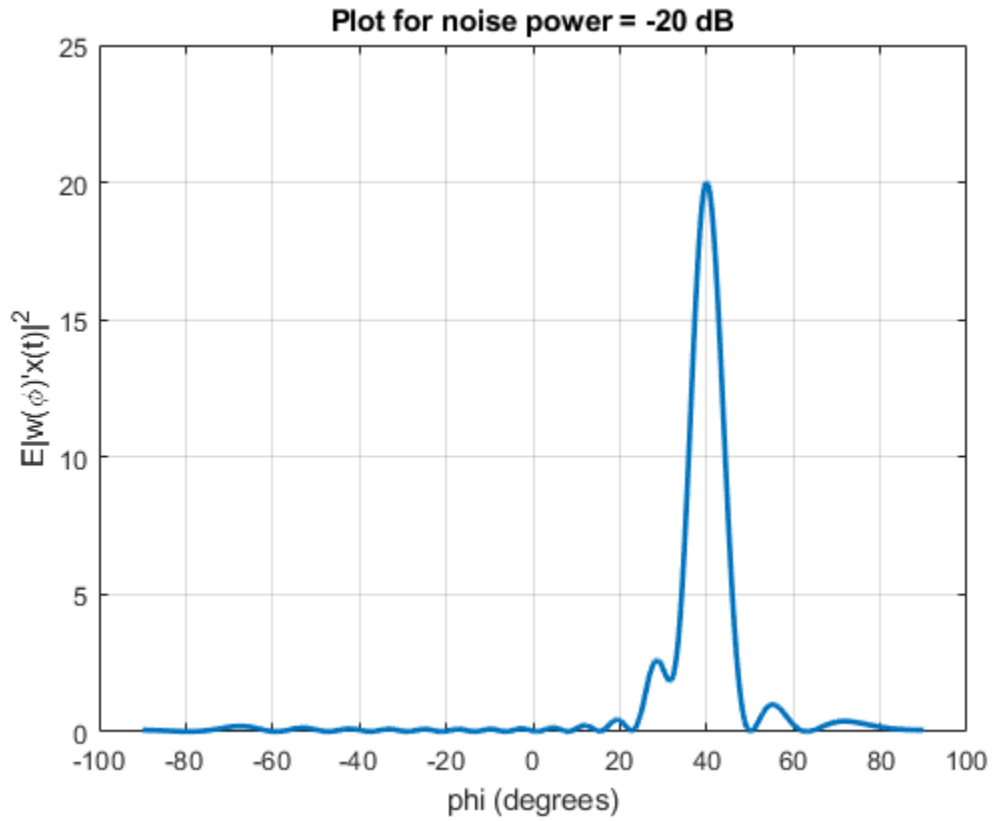
As you can see in the listed estimates, and the errors between the estimated phi2 and true phi2 values, they are relatively accurate estimates for the best phi2. Generally, sometimes the error can increase a little bit when the Noise Power (dB) is greater than 0. This procedure does give a perfect estimate for phi2 as you can see most have negligible error values.

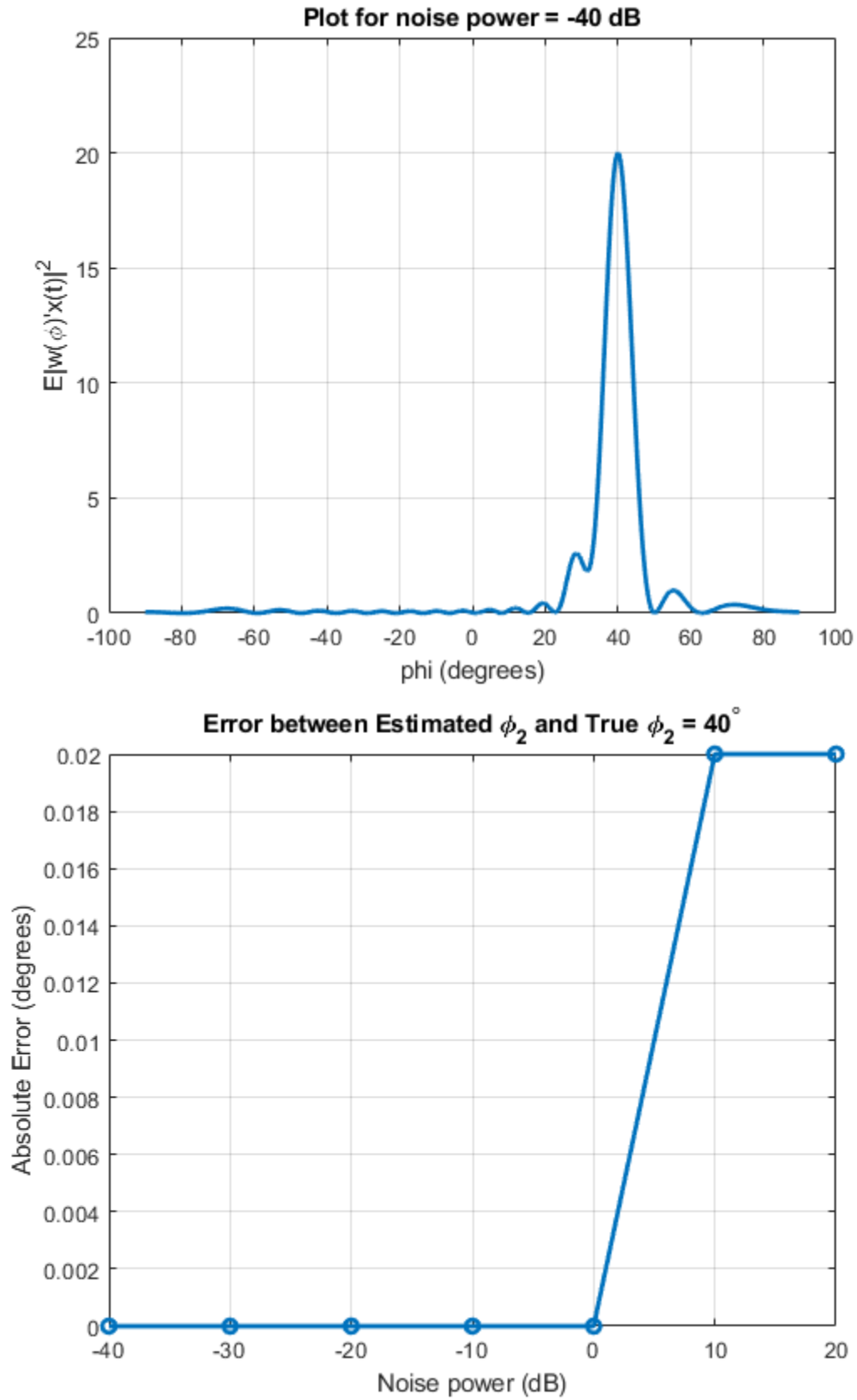
```
estimates =

    40.0200
    39.9800
    40.0000
    40.0000
    40.0000
    40.0000
    40.0000
```









(e) A variant of the minimum variance combiner: low-rank approximation

```
% Calculate SINR over SNR
SINR_dB_low_rank_mvc = zeros(size(SNR_dB)); % Preallocate SINR in dB

% Number of noise power values
num_vals = length(noise_power_dB);
for idx = 1:num_vals
    % Calculate noise power from SNR
    sigma2 = 1 / SNR(idx);

    phi2_hat = estimates(idx) * pi / 180;
    [low_rank_mvc, h2_hat] = generate_low_rank_MVC(N, T, delta_phi1, phi1, phi2_hat, zeta);

    % Initialize accumulators for SINR across trials
    SINR_accum = 0;
    for trial = 1:num_MC
        % Generate noise matrix for each trial
        n = generate_noise(N, T, sigma2);

        % Generate received signal
        x = calculate_received_signal_array(h1, h2, s1, s2, N, T, n);

        % Compute SINR for the MRC combiner
        SINR_trial_dB = calculate_SINR(low_rank_mvc, x, h1, h2, s1, s2, n, T);

        % Accumulate SINR results over trials
        SINR_accum = SINR_accum + 10^(SINR_trial_dB / 10); % Convert to linear scale for averaging
    end

    % Average SINR over trials and convert back to dB
    mean_SINR = SINR_accum / num_MC;
    SINR_dB_low_rank_mvc(idx) = 10 * log10(mean_SINR); % Convert back to dB
end
```

Plot of the SINR vs σ^2

```
plot_SINR_over_noise_power(SINR_dB_low_rank_mvc, noise_power_dB, "Low-Rank MVC Combiner");
```

Imperfect MVC and Low-Rank MVC Combiners

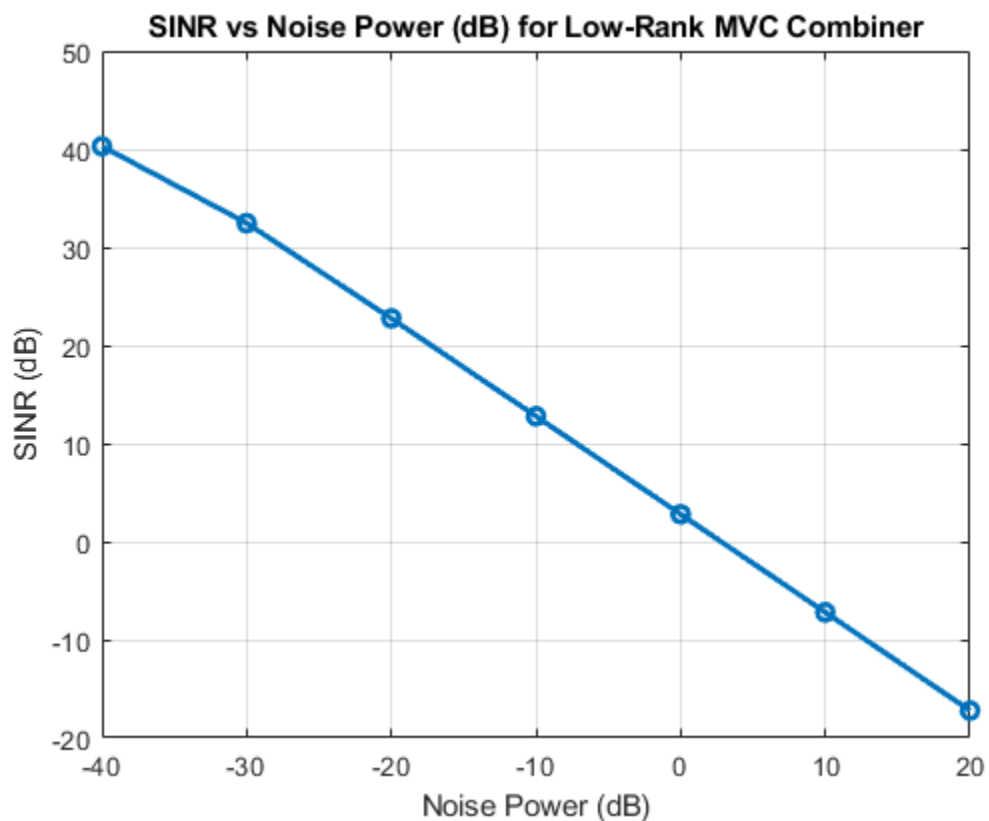
```
SINR_dB_list = {SINR_dB_imperfect_mvc, SINR_dB_low_rank_mvc};
combiner_names = {"Imperfect MVC Combiner", "Low-Rank MVC Combiner"};
plot_SINR_over_noise_power_compare(SINR_dB_list, noise_power_dB, combiner_names)
```

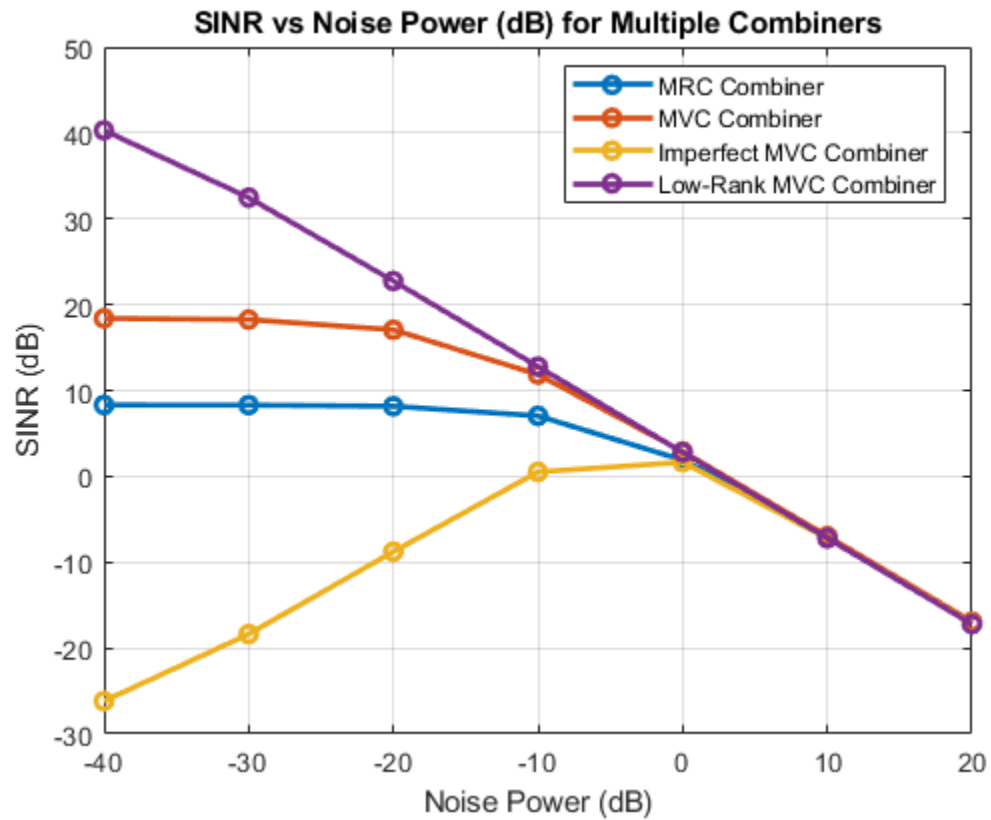
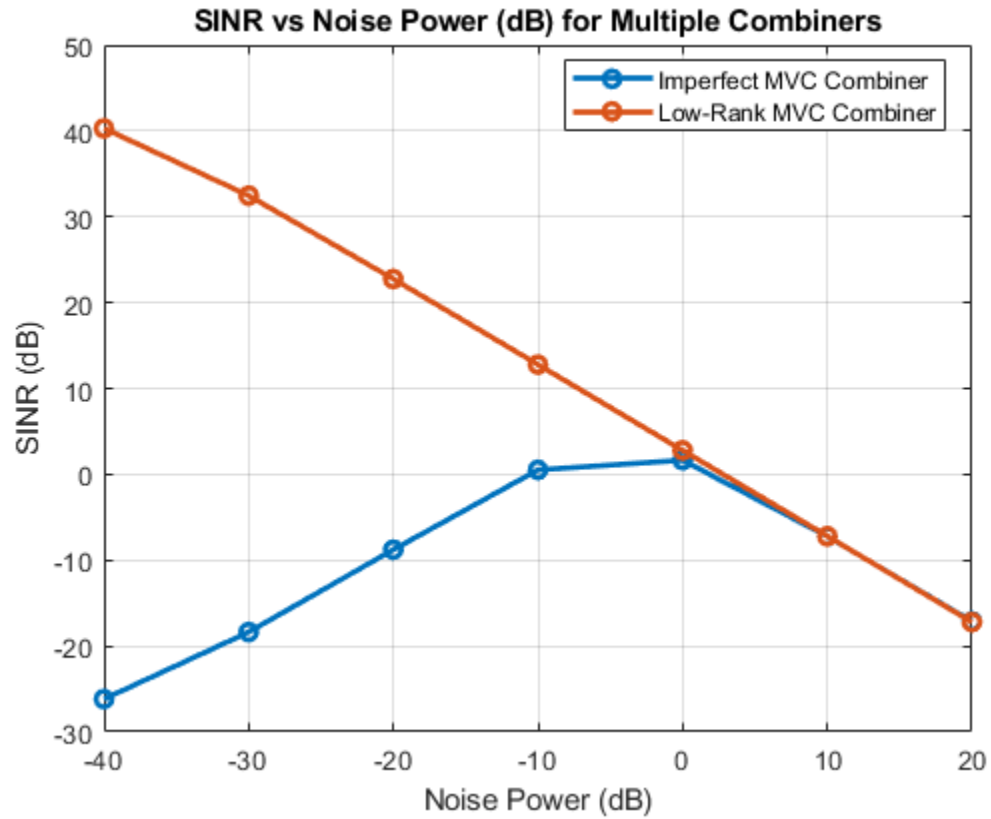
All combiners

```
SINR_dB_list = {SINR_dB_mrc, SINR_dB_mvc, SINR_dB_imperfect_mvc,  
SINR_dB_low_rank_mvc};  
combiner_names = {"MRC Combiner", "MVC Combiner", "Imperfect MVC Combiner",  
"Low-Rank MVC Combiner"};  
plot_SINR_over_noise_power_compare(SINR_dB_list, noise_power_dB,  
combiner_names)
```

Explanation

The Low-Rank MVC combiner, unlike the Imperfect MVC Combiner, starts out the highest and starts merging with the other combiners at approximately Noise Power = 0 dB. The Imperfect MVC Combiner starts out with a low SINR value and increases until it merges with the MRC and MVC combiners at Noise Power = 0 dB. The Low-Rank MVC combiner is more of a negative linear trend while the imperfect MVC trends up and then down. All of the combiners trend in the same direction once they merge at approximately Noise Power = 0 dB.





Functions

```
function h = calculate_ULA_response(N, phi)
    % Channel vectors
    h = 1/sqrt(N) * exp(-1j * pi * (0:N-1)' * sin(phi)); % N x 1 channel vector
end

function s = calculate_QAM_symbols(T, P)
    % Generate 4-QAM symbols
    s = sqrt(P) * (randi([0 1], T, 2) * 2 - 1 + 1j * (randi([0 1], T, 2) * 2 - 1)); % 4-QAM symbols
end

function X = calculate_received_signal_array(h1, h2, s1, s2, N, T, n)
    % Initialize the received signal matrix X
    X = zeros(N, T);

    % Loop through each time step and compute the received signal for each t
    for t = 1:T
        % Calculate the received signal at time t
        X(:, t) = h1 * s1(t) + h2 * s2(t) + n(:, t);
    end
end

function n = generate_noise(N, T, sigma2)
    % Generate NxT complex Gaussian noise with variance sigma2
    n_real = sqrt(sigma2/2) * randn(N, T); % Real part
    n_imag = sqrt(sigma2/2) * randn(N, T); % Imaginary part

    % Combine the real and imaginary parts to form complex noise
    n = n_real + 1i * n_imag;
end

function [s1_hat, target_signal, interference, noise] =
compute_combined_signal(w, x, h1, h2, s1, s2, n, T)
    % Initialize output variables
    s1_hat = zeros(1, T);
    target_signal = zeros(1, T);
    interference = zeros(1, T);
    noise = zeros(1, T);

    % Loop over each time step
    for t = 1:T
        % Define the target signal component
        target_signal(t) = w' * h1 * s1(t);

        % Define the interference component
        interference(t) = w' * h2 * s2(t);

        % Define the noise component
        noise(t) = w' * n(:, t);
    end
end
```

```
% Estimate the combined signal (s1_hat) as the sum of the target,
interference, and noise
s1_hat(t) = target_signal(t) + interference(t) + noise(t);
end
end

function SINR_dB = calculate_SINR(w, x, h1, h2, s1, s2, n, T)
[s1_hat, target_signal, interference, noise] =
compute_combined_signal(w, x, h1, h2, s1, s2, n, T);

% Initialize signal and interference+noise power accumulators
signal_power_sum = 0;
interference_plus_noise_power_sum = 0;

% Loop through each time step
for t = 1:T
    % Calculate target signal contribution
    signal_component = abs(target_signal(t))^2;

    % Calculate interference and noise contribution
    interference_plus_noise_component = abs(interference(t) +
noise(t))^2;

    % Accumulate power
    signal_power_sum = signal_power_sum + signal_component;
    interference_plus_noise_power_sum =
interference_plus_noise_power_sum + interference_plus_noise_component;
end

% Compute the average power of signal and interference+noise
avg_signal_power = signal_power_sum / T;
avg_interference_plus_noise_power = interference_plus_noise_power_sum /
T;

% Calculate the SINR (ratio of average signal power to
interference+noise power)
SINR = avg_signal_power / avg_interference_plus_noise_power;

% Convert to dB
SINR_dB = 10 * log10(SINR);
end

function w = generate_MRC_combiner(h1)
w = h1;
end

function w = generate_MVC_combiner(x, h, N, T)
% Initialize the autocorrelation matrix R
R = zeros(N, N);

% Summing over 1000 realizations to compute R
for i = 1:T
    % Accumulate the outer product of each received signal vector
```

```

        R = R + (x(:, i) * x(:, i)');
    end

    % Take the average
    R = R / T;

    % Compute the MVC combiner weights
    w = R \ h; % Equivalent to inv(R) * h1, but numerically more stable
end

function w = generate_imperfect_MVC(x, N, T, delta_phi, phi)
    h1_hat = (1 / sqrt(N)) * exp(-1j * pi * (0:N-1).' * sin(delta_phi + phi));
    w = generate_MVC_combiner(x, h1_hat, N, T);
end

function [w_phi, r_phi, phi2_hat] = estimate_phi2(x, N, T, phi_range)
    w_phi = zeros(N, length(phi_range));
    r_phi = zeros(1, length(phi_range));

    for idx = 1:length(phi_range)
        deg = phi_range(idx);

        phi = deg * pi / 180;

        % compute w(phi)
        w_phi(:, idx) = (1/sqrt(N)) * exp(-1j * pi * (0:N-1)' * sin(phi));

        R = 0;
        for i = 1:T
            R = R + abs(w_phi(:, idx))' * x(:, i)).^2;
        end
        R = R / T;
        % Take the average
        r_phi(idx) = abs(R); % expected power = E|w(phi)' * x(t)|^2
    end
    [~, max_idx] = max(r_phi); % Take absolute value to consider the magnitude
    phi2_hat = phi_range(max_idx); % Best angle estimate in DEGREES
end

function [w, h2_hat] = generate_low_rank_MVC(N, T, delta_phi1, phi1, phi2_hat, zeta) % make sure all phi values are in radians
    % h2_hat
    h2_hat = (1/sqrt(N)) * exp(-1j * pi * (0:N-1)' * sin(phi2_hat));

    % R = h2_hat * h2_hat' + zeta*I
    R = zeros(N, N);
    for i = 1:T
        % Accumulate the outer product of each received signal vector
        R = h2_hat * h2_hat';
    end
    R = R / T;
    R = R + zeta * eye(N);

```

```
% w = R(zeta) \ h1_hat
h1_hat = (1 / sqrt(N)) * exp(-1j * pi * (0:N-1).' * sin(delta_phi1 +
phi1));
w = R \ h1_hat;
end

function plot_SINR_over_noise_power(SINR_dB, noise_power_dB, combiner_name)
    plot_title = sprintf('SINR vs Noise Power (dB) for %s', combiner_name);

    figure;
    plot(noise_power_dB, SINR_dB, '-o', 'LineWidth', 2);
    xlabel('Noise Power (dB)');
    ylabel('SINR (dB)');
    title(plot_title);
    grid on;
end

function plot_SINR_over_noise_power_compare(SINR_dB_list, noise_power_dB,
combiner_names)
    % Check that the number of combiners matches the SINR data provided
    num_combiners = length(combiner_names);

    % Create a figure for the plot
    figure;

    % Plot each combiner's SINR data
    for i = 1:num_combiners
        plot(noise_power_dB, SINR_dB_list{i}, '-o', 'LineWidth', 2);
        hold on; % Retain the current plot when adding new plots
    end

    % Add labels and title
    xlabel('Noise Power (dB)');
    ylabel('SINR (dB)');
    title('SINR vs Noise Power (dB) for Multiple Combiners');

    % Add a legend for the combiners
    legend(combiner_names, 'Location', 'Best');

    % Display the grid
    grid on;
end
```

Published with MATLAB® R2024a