% Simulation Parameters

numUsers = 4;

numSubcarriers = 64;

snrRange = 0:1:10; % SNR in dB

infoBitsLength = 1e6; % Length of info bits per user

diversityLevels = [1, 2, 3, 4]; % Diversity levels (e.g., MRC paths)

fairnessFactor = 10; % Factor to control fairness

rng(1); % Seed for reproducibility

% Initialize result matrices

berResultsWF = zeros(length(snrRange), numUsers, length(diversityLevels)); % WF BER

berResultsRR = zeros(length(snrRange), numUsers, length(diversityLevels)); % RR BER

sumCapacityWF = zeros(length(snrRange), 1); % WF Sum Capacity

sumCapacityRR = zeros(length(snrRange), 1); % RR Sum Capacity

totalThroughputWF = zeros(length(snrRange), 1); % WF Throughput

totalThroughputRR = zeros(length(snrRange), 1); % RR Throughput

totalPowerWF = zeros(length(snrRange), numUsers); % Total power used by WF

totalPowerRR = zeros(length(snrRange), numUsers); % Total power used by RR

% Initialize fairness result matrices

fairnessIndexWF = zeros(length(snrRange), 1); % Fairness index for WF

fairnessIndexRR = zeros(length(snrRange), 1); % Fairness index for RR

% Main Simulation Loop

for snrIdx = 1:length(snrRange)

snr = snrRange(snrIdx);

noiseVar = 10^(-snr / 10); % Noise variance

% Random channel gains for each user and subcarrier

channelGains = abs(randn(numUsers, numSubcarriers));

%% Water-Filling Power Allocation

waterLevel = 1 / mean(channelGains(:));

allocatedPowerWF = zeros(numUsers, numSubcarriers);

for user = 1:numUsers

for sc = 1:numSubcarriers

if channelGains(user, sc) > 0

allocatedPowerWF(user, sc) = max(0, waterLevel - 1 / channelGains(user, sc));

end

end

end

%% Round Robin Power Allocation

allocatedPowerRR = ones(numUsers, numSubcarriers) \* fairnessFactor / numSubcarriers; % Equal power allocation with fairness

%% Sum Capacity Calculation

sumCapacityWF(snrIdx) = sum(log2(1 + (channelGains .\* allocatedPowerWF) / noiseVar), 'all');

sumCapacityRR(snrIdx) = sum(log2(1 + (channelGains .\* allocatedPowerRR) / noiseVar), 'all');

%% Total Power Usage Calculation

totalPowerWF(snrIdx, :) = sum(allocatedPowerWF, 2); % Total power used by WF for each user

totalPowerRR(snrIdx, :) = sum(allocatedPowerRR, 2); % Total power used by RR for each user

%% BER Calculation for WF and RR

for diversityIdx = 1:length(diversityLevels)

diversityOrder = diversityLevels(diversityIdx);

for user = 1:numUsers

% Generate random binary data

userData = randi([0 1], infoBitsLength, 1);

% BPSK Modulation

txSymbols = 1 - 2 \* userData;

% Add diversity effect

rxSymbolsWF = zeros(length(txSymbols), diversityOrder);

rxSymbolsRR = zeros(length(txSymbols), diversityOrder);

for diversitySample = 1:diversityOrder

% AWGN noise for WF

noiseWF = sqrt(noiseVar / 2) \* (randn(size(txSymbols)) + 1j \* randn(size(txSymbols)));

rxSymbolsWF(:, diversitySample) = txSymbols + noiseWF;

% AWGN noise for RR

noiseRR = sqrt(noiseVar / 2) \* (randn(size(txSymbols)) + 1j \* randn(size(txSymbols)));

rxSymbolsRR(:, diversitySample) = txSymbols + noiseRR;

end

% Diversity combining

combinedSymbolsWF = mean(rxSymbolsWF, 2);

combinedSymbolsRR = mean(rxSymbolsRR, 2);

% Decoding

receivedBitsWF = real(combinedSymbolsWF) < 0;

receivedBitsRR = real(combinedSymbolsRR) < 0;

% BER calculation

berResultsWF(snrIdx, user, diversityIdx) = sum(receivedBitsWF ~= userData) / infoBitsLength;

berResultsRR(snrIdx, user, diversityIdx) = sum(receivedBitsRR ~= userData) / infoBitsLength;

end

end

%% Throughput Calculation (using the last diversity level)

totalThroughputWF(snrIdx) = sum((1 - berResultsWF(snrIdx, :, end)) \* infoBitsLength);

totalThroughputRR(snrIdx) = sum((1 - berResultsRR(snrIdx, :, end)) \* infoBitsLength);

%% Fairness Index Calculation

% Calculate throughput for each user

userThroughputWF = zeros(numUsers, 1);

userThroughputRR = zeros(numUsers, 1);

for user = 1:numUsers

% For WF, calculate user throughput

userThroughputWF(user) = sum(log2(1 + (channelGains(user, :) .\* allocatedPowerWF(user, :)) / noiseVar));

% For RR, calculate user throughput

userThroughputRR(user) = sum(log2(1 + (channelGains(user, :) .\* allocatedPowerRR(user, :)) / noiseVar));

end

% Calculate Jain's Fairness Index for WF and RR

fairnessIndexWF(snrIdx) = (sum(userThroughputWF)^2) / (numUsers \* sum(userThroughputWF.^2));

fairnessIndexRR(snrIdx) = (sum(userThroughputRR)^2) / (numUsers \* sum(userThroughputRR.^2));

end

%% Display Results (BER, Throughput, Fairness, Sum Capacity)

fprintf('\nResults Summary:\n');

fprintf('SNR (dB) | WF BER (All Users) | RR BER (All Users) | WF Throughput | RR Throughput | WF Fairness | RR Fairness | WF Sum Capacity | RR Sum Capacity\n');

fprintf('------------------------------------------------------------------------------------------------------\n');

for snrIdx = 1:length(snrRange)

% Average BER for all users at the final diversity level

avgBERWF = mean(berResultsWF(snrIdx, :, end));

avgBERRR = mean(berResultsRR(snrIdx, :, end));

% Print the summary for each SNR

fprintf('%8.2f | %18.5f | %18.5f | %14.2f | %14.2f | %12.5f | %12.5f | %16.4f | %16.4f\n', ...

snrRange(snrIdx), avgBERWF, avgBERRR, totalThroughputWF(snrIdx), totalThroughputRR(snrIdx), ...

fairnessIndexWF(snrIdx), fairnessIndexRR(snrIdx), sumCapacityWF(snrIdx), sumCapacityRR(snrIdx));

end

%% Combined BER Graph for All Users

figure;

hold on;

for user = 1:numUsers

plot(snrRange, berResultsWF(:, user, end), 'LineWidth', 1.5, 'DisplayName', sprintf('WF User %d', user));

plot(snrRange, berResultsRR(:, user, end), '--', 'LineWidth', 1.5, 'DisplayName', sprintf('RR User %d', user));

end

set(gca, 'YScale', 'log'); % Use logarithmic scale for BER

title('Combined BER Comparison for All Users');

xlabel('SNR (dB)');

ylabel('Bit Error Rate (BER)');

legend('show');

grid on;

%% Separate BER Graphs for Each User

for user = 1:numUsers

figure;

plot(snrRange, berResultsWF(:, user, end), 'r-o', 'LineWidth', 1.5, 'DisplayName', 'WF');

hold on;

plot(snrRange, berResultsRR(:, user, end), 'b-x', 'LineWidth', 1.5, 'DisplayName', 'RR');

set(gca, 'YScale', 'log'); % Use logarithmic scale for BER

title(sprintf('BER Comparison for User %d', user));

xlabel('SNR (dB)');

ylabel('Bit Error Rate (BER)');

legend('show');

grid on;

end

%% Sum Capacity Comparison

figure;

plot(snrRange, sumCapacityWF, 'r-o', 'LineWidth', 2, 'DisplayName', 'Water-Filling');

hold on;

plot(snrRange, sumCapacityRR, 'b-x', 'LineWidth', 2, 'DisplayName', 'Round Robin');

xlabel('SNR (dB)');

ylabel('Sum Capacity (bps/Hz)');

title('Sum Capacity Comparison');

legend('show');

grid on;

%% Throughput Comparison

figure;

plot(snrRange, totalThroughputWF, 'k-o', 'LineWidth', 2, 'DisplayName', 'Water-Filling');

hold on;

plot(snrRange, totalThroughputRR, 'g-x', 'LineWidth', 2, 'DisplayName', 'Round Robin');

xlabel('SNR (dB)');

ylabel('Total Throughput (bits)');

title('Throughput Comparison');

legend('show');

grid on;

%% Fairness Index Comparison

figure;

plot(snrRange, fairnessIndexWF, 'r-o', 'LineWidth', 2, 'DisplayName', 'Water-Filling');

hold on;

plot(snrRange, fairnessIndexRR, 'b-x', 'LineWidth', 2, 'DisplayName', 'Round Robin');

xlabel('SNR (dB)');

ylabel('Fairness Index');

title('Fairness Index Comparison');

legend('show');

grid on;

hold off;

% Total Power Allocation Graph

figure;

bar([totalPowerWF, totalPowerRR]);

set(gca, 'XTickLabel', arrayfun(@(x) sprintf('%d dB', x), snrRange, 'UniformOutput', false));

xlabel('SNR (dB)');

ylabel('Total Power (Watts)');

title('Total Power Allocation: WF vs RR');

legend({'Water-Filling', 'Round Robin'});

grid on;