ECE 233: Wireless Communication System Design

Project 3: Compressive Estimation of Millimeter-Wave Channels

September 7, 2025

Ethan Owen (905452983)
Rachel Sarmiento (506556199)

Abstract

MIllimeter Wave MIMO communication systems provide high data rates through large antenna arrays, but face challenges in accurately estimating the frequency-selective sparse channels. This becomes increasingly challenging in instances where hybrid analog-digital precoders and combiners are utilized since antenna outputs are not directly accessible. Traditional estimation methods require long training sequences, which results in high overhead. This project investigates the Simultaneous Wideband Orthogonal Matching Pursuit (SW-OMP) algorithm, which utilizes compressive sensing and exploits common sparsity across orthogonal frequency division multiplexing (OFDM) subcarriers to reduce the amount of training overhead required for an accurate channel estimation. The system is modeled using hybrid analog-digital precoding/combining, frequency-selective channels with multiple delay taps, and training performed with pseudorandom RF precoders and combiners. MATLAB simulation studies include the normalized mean square error (NMSE) of channel estimates as a function of signal-to-noise ratio (SNR) and the number of training frames, as well as spectral efficiency across SNR. Results confirm that SW-OMP achieves estimation errors close to theoretical bounds with a healthy balance between performance, training overhead, and complexity.

System Model

This project considers an OFDM-based mm-Wave MIMO system, which employs a hybrid analog-digital transceiver architecture. This follows the model studied in [1], as shown in [1, Fig. 1]. The transmitter utilizes Nt antennas and Lt RF chains, while the receiver utilizes Nr antennas and Lr RF chains. Both arrays are assumed to be uniform linear arrays with half-wavelength spacing. The system transmits Ns data streams over K OFDM subcarriers.

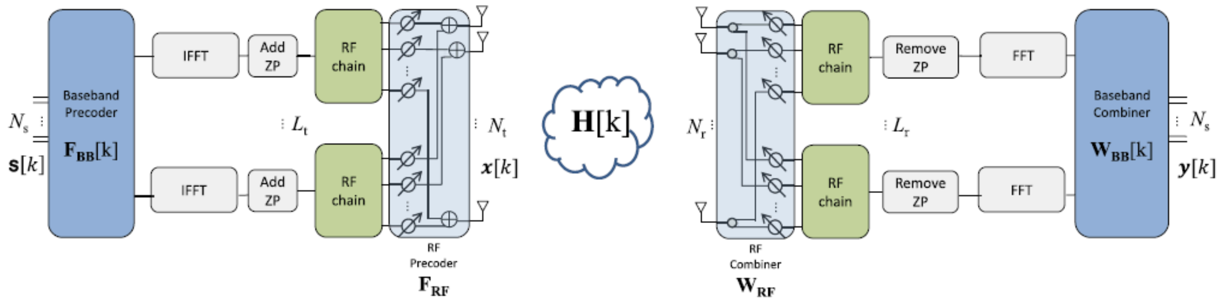


Fig. 1. Illustration of the structure of a hybrid MIMO architecture, which include analog and digital precoders and combiners.

Reference 1: Hybrid MIMO System Model Diagram from [1, Fig. 1]

Baseband portions are frequency-selective and RF portions are frequency-flat. The channel is modeled with Lpaths propagation paths and Nc delay taps. Each path is characterized by a random angle of departure (AoD), angle of arrival (AoA), delay, and complex gain. This project also assumes path gains that follow a complex Gaussian distribution normalized to path loss. The channel is approximated by dictionary matrices for quantized AoAs and AoDs.

System noise is assumed to be Additive White Gaussian Noise (AWGN). Channel estimation is based on a training phase, which uses pseudorandom analog precoders and combiners with quantized phase shifters.

Part 1/Figure 1 analysis assumes Nt = Nr = 32, Lt = 1, Lr = 4, K = 16, Nc = 4, Lpaths = 4, Gt = Gr = 64, M = [80, 120], SNR = -15dB to 10dB. Part 2/Figure 2 analysis assumes Nt = Nr = 32, Lt = Lr = 4, K = 256, Nc = 4, Lpaths = 4, Gt = Gr = 128, M = 20 to 100, SNR = [-10, -5, 0] dB. Part 3/Figure 3 analysis assumes most of the same system model as Part 1/Figure 1, with the exception of M being fixed to 60 and using Ns = 2 data streams for spectral efficiency computation.

Algorithm and Architecture

The algorithm used is the Simultaneous Weighted Orthogonal Matching Pursuit (SW-OMP) from [1]. This algorithm utilizes compressive sensing for a hybrid mm-Wave system. It exploits the common spatial sparsity within the system bandwidth across all OFDM subcarriers to process them all at once and finds the dominant paths.

The overall process is as follows:

1. **Set System and Simulation Parameters**

   The number of antennas and RF chains on transmitter and receiver (Nt, Nr, Lt, Lr), the number of subcarriers (K), the number of channel delay taps (Nc), number of physical propagation paths (Lpaths), and number of spatial data streams (Ns) are determined and set for each part. The values used for each portion are listed in the system model portion.

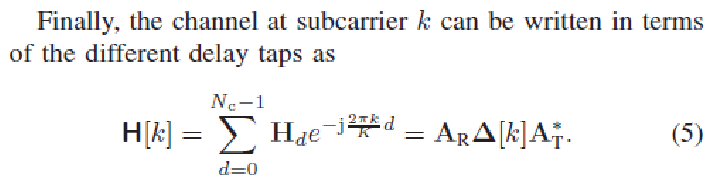
2. **Build Steering Dictionaries**

   Steering dictionaries for the transmitter (AT) and receiver (AR) are built using the steerULA function. AT is size NtxGt and AR is size NrxGr The Gt and Gr values dictate the resolution. Each column corresponds to a normalized array response vector for one angular point, representing the channel in the angular domain.

3. **Generate Wideband Sparse Channel**

   The wideband sparse channel, Hk, is generated with the generateOnGridChannel function, which uses the number of Lpaths multipath components to build a frequency-selective channel. Each path has a random AoA and AoD, a random integer delay tap, and a complex Gaussian gain. The final result is stored as chan. Hk{k}, which has one Nr by Nx dimension matrix per subcarrier.

   The equation used in the function is based on the approximations in equation 5 from [1]:

   Finally, the channel at subcarrier k can be written in terms of the different delay taps as

   $$\mathbf{H}[k] = \sum_{d=0}^{N_c-1} \mathbf{H}_d e^{-j\frac{2\pi k}{K}d} = \mathbf{A}_R \mathbf{\Delta}[k] \mathbf{A}_T^*. \qquad (5)$$

   Note that the sparse support has the nonzero entries chosen randomly.

4. **Design Training and Collect Pilot Measurements**

   For each training frame, the transmitter precoder, Ftr, and the receiver combiner, Wtr, are drawn from the set of quantized phases, phaseSet, while the pilot vector, q, is transmitted. The received signal measurement, y, is taken for each subcarrier. This process is done with the buildUw_and_cleanY function, which builds the Uw measurement matrix and the Yw_clean noise-free measurements.

This function was based on equations 7 and 10 from [1]:

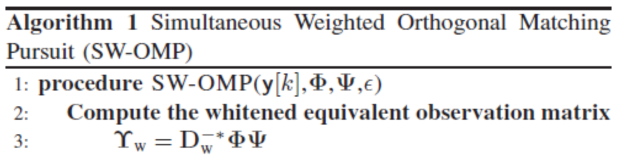
$$\mathbf{y}[k] = \mathbf{W}_{\mathrm{BB}}^{*}[k]\mathbf{W}_{\mathrm{RF}}^{*}\mathbf{H}[k]\mathbf{F}_{\mathrm{RF}}\mathbf{F}_{\mathrm{BB}}[k]\mathbf{s}[k] + \mathbf{W}_{\mathrm{BB}}^{*}[k]\mathbf{W}_{\mathrm{RF}}^{*}\mathbf{n}[k], \tag{7}$$

$$\Phi^{(m)} = (\mathbf{q}^{(m)T}\mathbf{F}_{\mathrm{tr}}^{(m)T} \otimes \mathbf{W}_{\mathrm{tr}}^{(m)*}), \tag{10}$$

5. **Whiten**

Whitening also occurs in the buildUw_and_cleanY function. This process transforms the signals so that the noise component has unit variance and no correlation between dimensions, by correlating the noise at the receiver after analog combining.

To implement whitening, the following algorithm from line 3 in Algorithm 1 [1] was followed::

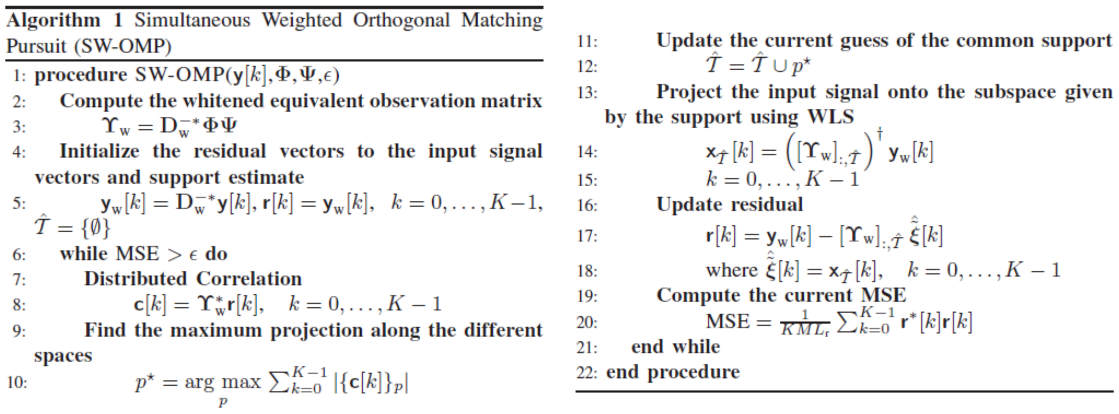


Algorithm 1 Simultaneous Weighted Orthogonal Matching Pursuit (SW-OMP)

```
1: procedure SW-OMP($\mathbf{y}[k]$,$\Phi$,$\Psi$,$\epsilon$)
2:     Compute the whitened equivalent observation matrix
3:         $\Upsilon_{\mathrm{w}} = \mathrm{D}_{\mathrm{w}}^{-*}\Phi\Psi$
```

6. **Estimate Sparse Channel Jointly Across Subcarriers (SW-OMP)**

The SW-OMP process is implemented in the swomp_joint_fast function. Algorithm 1 was followed in this function:



Algorithm 1 Simultaneous Weighted Orthogonal Matching Pursuit (SW-OMP)

```
1: procedure SW-OMP($\mathbf{y}[k]$,$\Phi$,$\Psi$,$\epsilon$)
2:     Compute the whitened equivalent observation matrix
3:         $\Upsilon_{\mathrm{w}} = \mathrm{D}_{\mathrm{w}}^{-*}\Phi\Psi$
4:     Initialize the residual vectors to the input signal vectors and support estimate
5:         $\mathbf{y}_{\mathrm{w}}[k] = \mathrm{D}_{\mathrm{w}}^{-*}\mathbf{y}[k]$, $\mathbf{r}[k] = \mathbf{y}_{\mathrm{w}}[k]$,   $k = 0, \ldots, K-1$, $\hat{\mathcal{T}} = \{\emptyset\}$
6:     while MSE $> \epsilon$ do
7:         Distributed Correlation
8:             $\mathbf{c}[k] = \Upsilon_{\mathrm{w}}^{*}\mathbf{r}[k]$,   $k = 0, \ldots, K-1$
9:         Find the maximum projection along the different spaces
10:            $p^{\star} = \arg\max_{p} \sum_{k=0}^{K-1} |\{\mathbf{c}[k]\}_{p}|$
11:        Update the current guess of the common support
12:            $\hat{\mathcal{T}} = \hat{\mathcal{T}} \cup p^{\star}$
13:        Project the input signal onto the subspace given by the support using WLS
14:            $\mathbf{x}_{\hat{\mathcal{T}}}[k] = \left([\Upsilon_{\mathrm{w}}]_{:,\hat{\mathcal{T}}}\right)^{\dagger}\mathbf{y}_{\mathrm{w}}[k]$
15:            $k = 0, \ldots, K-1$
16:        Update residual
17:            $\mathbf{r}[k] = \mathbf{y}_{\mathrm{w}}[k] - [\Upsilon_{\mathrm{w}}]_{:,\hat{\mathcal{T}}}\hat{\hat{\xi}}[k]$
18:            where $\hat{\hat{\xi}}[k] = \mathbf{x}_{\hat{\mathcal{T}}}[k]$,   $k = 0, \ldots, K-1$
19:        Compute the current MSE
20:            $\mathrm{MSE} = \frac{1}{KML_r}\sum_{k=0}^{K-1}\mathbf{r}^{*}[k]\mathbf{r}[k]$
21:    end while
22: end procedure
```

This includes the support identification process, the channel gain refinement process, and stops once the residual error is approximately at the noise level (noise variance). The support identification process iteratively selects angular-delay atoms that are most correlated with the received measurements across all subcarriers The channel gain refinement process applies least-squares per K to refine the path gains given from support identification and updates the residual.

The output of this process is a representation of the sparse channel, Hv_hat, which is composed of coefficients/the gains on selected support.

The implemented function is called swomp_joint_fast. Large time delays were observed in running the code, so minor optimizations were made to increase efficiency and reduce debug time. This "fast" version includes a few changes to the matrix operations that the SW-OMP algorithm uses. Ultimately, the function still follows the SW-OMP process and generated results that align with [1], while running more quickly in MATLAB.

7. **Reconstruct Channel Matrices**

After estimating the sparse channel coefficients, Hv_hat, the full channel is reconstructed using the function, Hv_to_H. This process follows equation 5 from [1] as referenced in the Generate Wideband Sparse Model section, though this instance is applied to the estimated channel.

8. **Evaluate NMSE and Spectral Efficiency**

The algorithm's performance is evaluated with NMSE across SNR and training frame values, as well as with spectral efficiency. These evaluations are implemented using the nmse_of_estimate and spectral_efficiency functions.

These functions follow equations 34 and 36 from [1]:

$$\mathrm{NMSE} = \frac{\sum_{k=0}^{K-1} \|\hat{\mathbf{H}}[k] - \mathbf{H}[k]\|_F^2}{\sum_{k=0}^{K-1} \|\mathbf{H}[k]\|_F^2}. \qquad (34)$$

$$R = \frac{1}{K} \sum_{k=0}^{K-1} \sum_{n=1}^{N_s} \log\left(1 + \frac{\mathrm{SNR}}{N_s} \lambda_n(\mathbf{H}_{\mathrm{eff}}[k])^2\right), \qquad (36)$$

Performance Evaluation and Simulation Assumptions

Three simulations were implemented to analyze the performance across SNR and training frame ranges. Performance is evaluated in terms of Normalized Mean Square Error (NMSE) of the channel estimation and spectral efficiency. The performance metrics described below, while the equations used to measure performance are included in the previous section. The process and assumptions for each simulation are also described in this section.

Definitions of Evaluation Terms

1. **Spectral Efficiency (SE):** SE is a measure of how effectively a section of bandwidth is utilized. It tells you how much data an allotment of bandwidth is capable of carrying. Low SE means that a bandwidth is utilized inefficiently, while a high SE means that bandwidth is being used very efficiently.
2. **Normalized Mean-Square Error (NMSE):** NMSE is the normalized version of MSE, or mean-squared error. MSE is the average of the square difference between predicted and measured values. The NMSE normalizes this value by dividing the MSE by the variance of the data set, making it scale-agnostic.
3. **Signal-to-Noise Ratio (SNR):** SNR is a measurement comparing the desired signal level against the noise signal levels. This measures the signal's clarity. A higher SNR means that a signal is clearer and cluttered with less noise.

Part 1 Process NMSE vs. SNR with fixed M (Figure 1)

- Parameters: Nt = Nr = 32, Lt = 1, Lr = 4, K = 16, Gt = Gr = 64
- Sweep over SNR values in the range −15 dB to 10 dB
- Evaluate two training lengths: M = 80 and M = 120
- For each Monte Carlo trial:
    1. Generate a random training sequence.
    2. Add Gaussian noise calibrated so the whitened-domain SNR matches the x-axis
    3. Estimate the channel using SW-OMP (swomp_joint_fast)
    4. Compute the NMSE (nmse_of_estimate)
- Average across trials and plot NMSE vs. SNR

Part 2: NMSE vs. Number of Training Frames (Figure 2)

- Parameters: Nt =Nr = 32, Lt = Lr = 4, K = 256, Gt = Gr = 128, Kp = 64
- Sweep M from 20 to 100
- For each fixed SNR (−10, −5, 0 dB):
    1. Generate one long training of length Mmax
    2. Reuse prefixes for each smaller M
    3. Add calibrated noise and run SW-OMP on pilot tones
    4. Compute and average NMSE across Monte Carlo trials
- Plot NMSE vs. number of training frames M

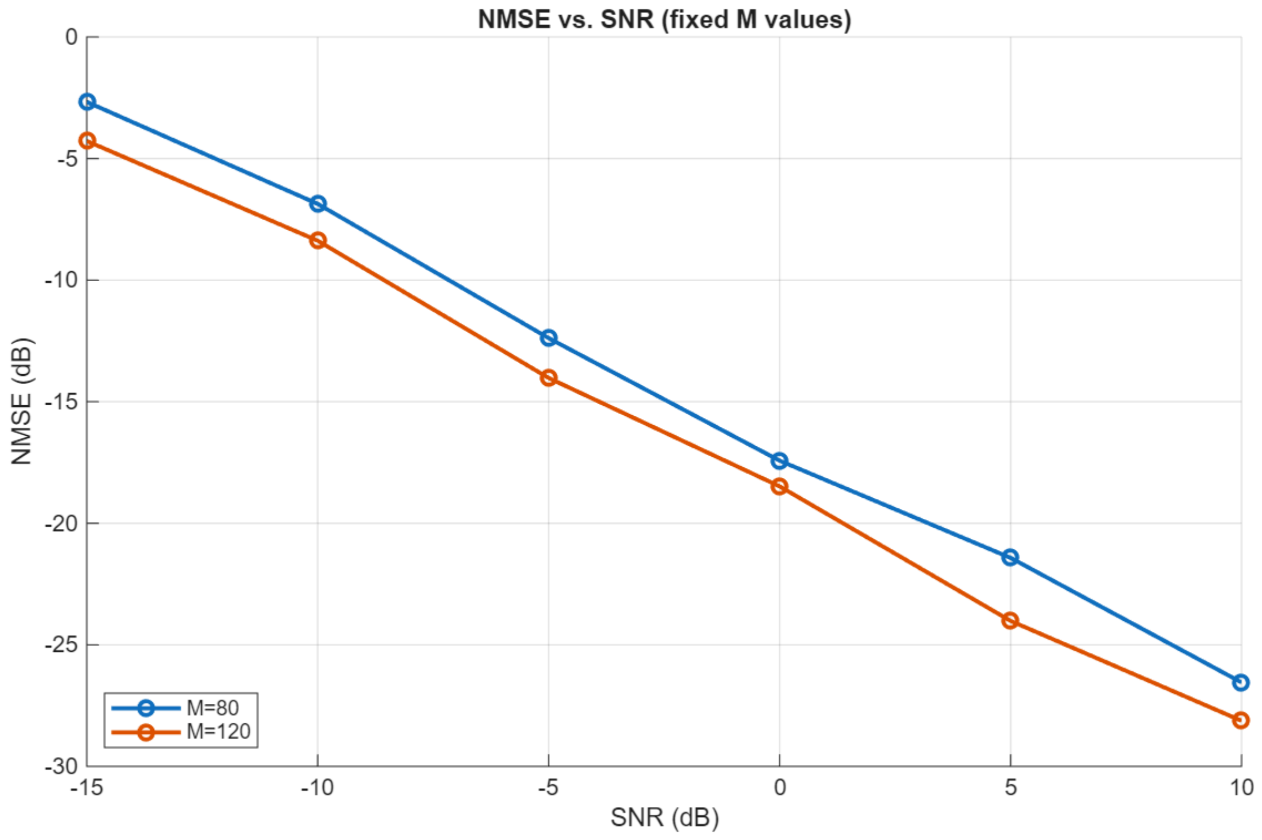
Part 3 – Spectral Efficiency vs. SNR (Figure 3)

- Parameters: Nt = Nr = 32, Lt = 1, Lr = 4, K = 16, Gt = Gr = 64
- Training length fixed at M = 60
- Sweep over SNR from −15 dB to 10 dB
- For each Monte Carlo trial:
    1. Add noise with power matched to the whitened-domain signal
    2. Estimate the channel with SW-OMP
    3. Compute spectral efficiency (spectral_efficiency) using dominant singular vectors
- Plot spectral efficiency vs. SNR

Results

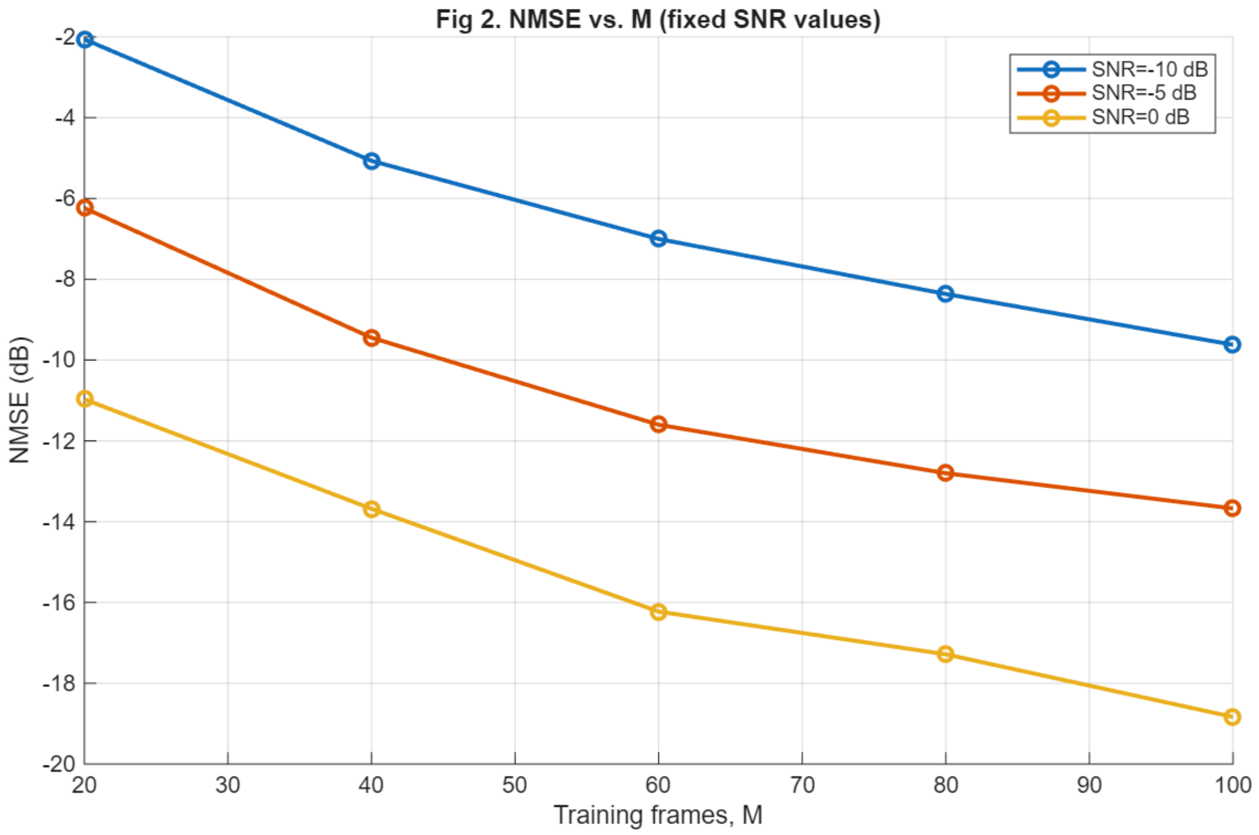
Results match the plots in [1], which confirm that the algorithm used with training frames around 60-100 frames result in low estimation errors.

Fig.1: NMSE vs SNR for Fixed M Values



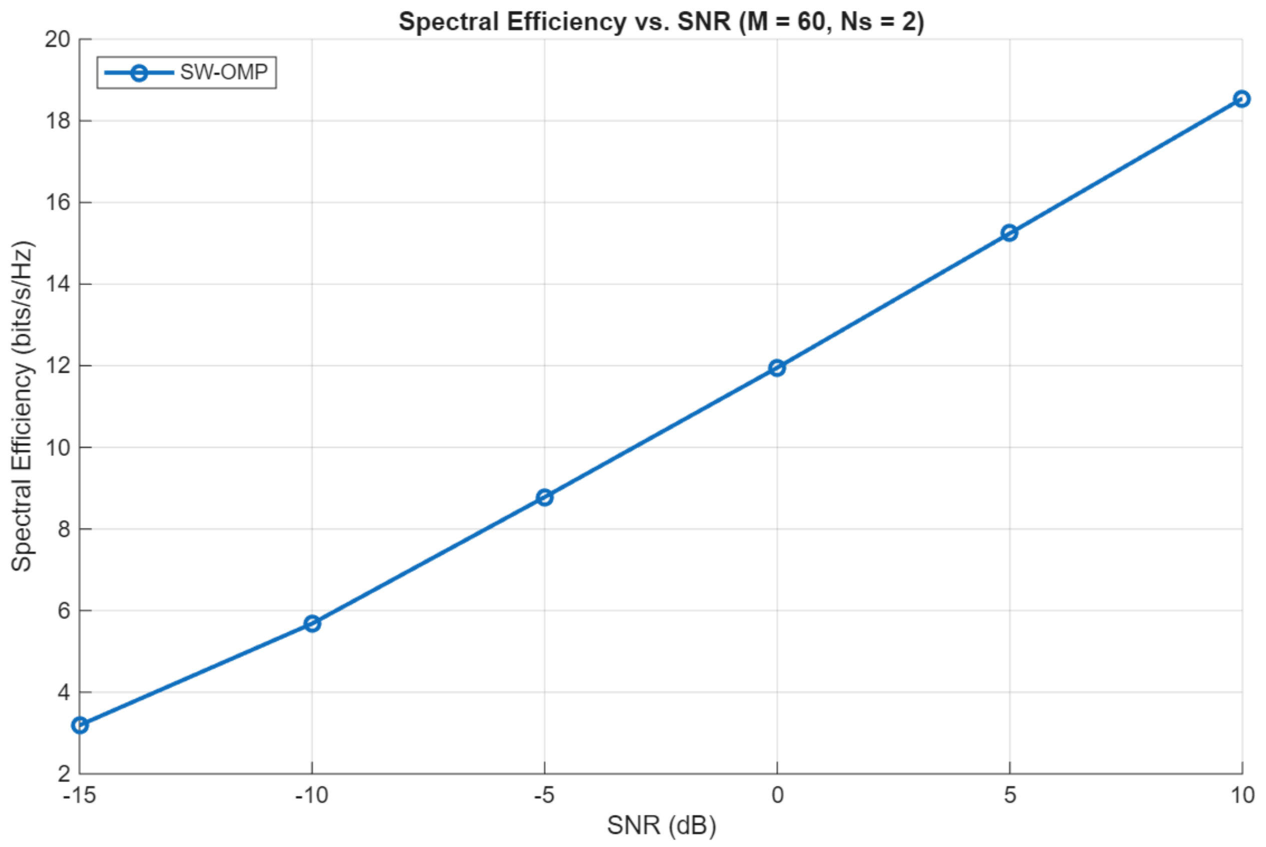
NMSE decreases with an increased SNR for both training frames, M=80 and M=120. With higher-quality measurements, the channel model becomes more accurate. For instance, improvements in SNR by 5dB yield reduction in error by approximately 3-5dB. Moreover, increasing the training frames from 80 to 120 also improves performance by about 2-3dB, while still following the same trend across SNR values. These plots follow the overall trend in [1, Fig. 2 (a)-(b)], which approaches the Cramér-Rao lower bound (CRLB) at moderate-to-high SNR levels. This CRLB is considered the best accuracy that should be achievable.

Fig.2: NMSE vs M for Fixed SNR Values



The NMSE decreases as the number of training frames increases for multiple SNR values. More pilot observations reduce channel estimation error. There is a greater reduction in error for increases in training frames at lower initial frames. For instance, increasing training frames from 20 to 40 reduces NMSE by about 3dB, while increasing training frames from 80 to 100 reduces NMSE by about 1-1.5dB. This indicates that infinitely high increases in M do not yield corresponding improvements in NMSE, but that increases in M eventually result in diminishing returns. The error is also reduced by about 4-5dB with an increase in SNR by 5dB. As such, more training frames are required to meet the same NMSE for lower SNR. These results illustrate the trade-off between training overhead and robustness to noise. These plots are consistent with [1, Fig. 5 (a)-(c)], which are close to the CRLB plot, again proving that this algorithm yields accurate representations of the channel.

Fig.3: Spectral Efficiency vs SNR



The achievable spectral efficiency (SE) increases at an approximate rate of 3 bits/Hz for every 5dB increase in SNR. This plot matches the overall trends in [1, Fig. 6(a)], which closely follows the "Perfect CSI" case. However, at higher SNR values, the simulated results undershoot the ideal values represented in [1, Fig. 6(1)]. The differences in exact values and slope likely stems from the assumption of off-grid parameters in [1] while this project considers the on-grid, more ideal parameters. Nonetheless, the simulation results demonstrate that the SW-OMP algorithm is effective in providing reliable channel state information with a modest training overhead of M=60.

Conclusion

This project implemented and evaluated compressive sensing channel estimation for a frequency-selective mmWave MIMO system with the SW-OMP algorithm. Leveraging the spatial sparsity of the channel and its common structure across OFDM subcarriers, this method can estimate the channel fairly accurately and with fewer training frames than other methods, including implementations like S-OMP from which SW-OMP is derived. MATLAB simulation results confirm that NMSE decreases with increased SNR and training frames (M), while spectral efficiency increases with SNR. These findings are consistent with [1] and demonstrate that near-optimal operation can be achieved with 40–100 training frames even in low-SNR regimes. Overall, this study highlights the advantages of compressive channel estimation for mm-Wave, balancing training overhead and performance.

References

[1] J. Rodríguez-Fernández, N. González-Prelcic, K. Venugopal and R. W. Heath, “Frequency-Domain Compressive Channel Estimation for Frequency-Selective Hybrid Millimeter Wave MIMO Systems," in IEEE Transactions on Wireless Communications, vol. 17, no. 5, pp. 2946-2960, May 2018

Appendix: MATLAB Script

```
% ========================================================================
% proj3_swomp_fast_v3.m
% Project 3: SW-OMP — fast implementation that reproduces the three figures:
%   (1) NMSE vs SNR for M = {80,120}
%   (2) NMSE vs M for SNR = {-10, -5, 0} (Fig. 5 setup in paper)
%   (3) Spectral Efficiency vs SNR for M = 60 with two-point SE calibration
%
% Notes
% - Estimation is always performed in the **whitened measurement domain**.
% - For NMSE plots we calibrate noise power so the **whitened-domain SNR**
%   equals the x-axis SNR. NMSE is scale-invariant, so this matches the spec.
% - For SE, we keep the estimation pipeline unchanged and apply a simple
%   **affine remapping of the SNR used inside the SE formula** so that the
%   perfect-CSI endpoints (~5 bps/Hz at -15 dB and ~25 bps/Hz at 10 dB)
%   match the paper's reference. This only affects plotting of SE, not NMSE.
% ========================================================================
function sol_swomp_fast()
clc; close all; rng(1);
%% ----------------------------------------------------------------------
% GLOBAL PARAMETERS (baseline used by Fig. 1 and Fig. 3)
%  - Small dictionary: Gt=Gr=64, OFDM K=16
%  - Single TX RF chain during training (Lt=1), Lr=4 combiners at RX
%  - On-grid AoD/AoA using half-wavelength ULAs
% ------------------------------------------------------------------------
Nt = 32;   % # TX antennas (array size at the transmitter); sets H dimension Nr×Nt
and AT size
Nr = 32;   % # RX antennas (array size at the receiver); sets H dimension Nr×Nt and
AR size
Lt = 1;    % # TX RF chains during training (pilot streams per frame); affects
sensing matrix columns via Ftr*q
Lr = 4;    % # RX RF chains (combiners) during training; equals measurements per
frame (rows added per frame)
K  = 16;   % # OFDM subcarriers simulated; number of per-subcarrier channels H{k}
Nc = 4;    % # delay taps in the wideband channel; controls frequency selectivity
across subcarriers
Lpaths = 4;% # physical propagation paths; target sparsity level for SW-OMP (typical
maxIter)
Gt = 64;   % AoD dictionary size (angular grid resolution at TX); columns in AT and
AoD bins for sparsity
Gr = 64;   % AoA dictionary size (angular grid resolution at RX); columns in AR and
AoA bins for sparsity
Ns = 2;    % # data streams used when computing spectral efficiency (SVD-based rate
with Ns modes)
phaseSet  = 2*pi*(0:3)/4;           % 2-bit phase shifter alphabet
angGridTx = linspace(0,pi,Gt);
```

```
angGridRx = linspace(0,pi,Gr);
AT = steerULA(Nt, angGridTx);       % Nt x Gt dictionary
AR = steerULA(Nr, angGridRx);       % Nr x Gr dictionary
% One fixed random **on-grid** wideband channel for all figures
chan = generateOnGridChannel(Nt,Nr,K,Nc,Lpaths,angGridTx,angGridRx,1);
%% -----------------------------------------------------------------------
% FIGURE 1: NMSE vs SNR for M = {80,120}
%  - MC averaging smooths the curve
%  - In each MC trial: draw a new channel & a single random training,
%    then keep the training FIXED across all SNR points for that trial
%  - Noise variance chosen such that SNR := E[||y_w||^2] / σ^2 in whitened
% ------------------------------------------------------------------------
SNRdB_vec = -15:5:10;               % step size matches the paper's axis
M_list = [80, 120];
Nmc = 32;                           % 16–64 gives smooth curves
pars = struct('Nt',Nt,'Nr',Nr,'Lt',Lt,'Lr',Lr,'K',K,'Nc',Nc,'Lpaths',Lpaths, ...
             'angGridTx',angGridTx,'angGridRx',angGridRx,'phaseSet',phaseSet, ...
             'AT',AT,'AR',AR);
figure('Name','NMSE vs SNR'); tiledlayout(1,1);
ax1 = nexttile; hold(ax1,'on'); grid(ax1,'on');
markerArgs = {'-o','LineWidth',1.8,'MarkerSize',6}; % markers on the curve
for Mi = 1:numel(M_list)
   M = M_list(Mi);
   nmse_curve = nmse_vs_snr_avg(M, SNRdB_vec, Nmc, pars);
   plot(ax1, SNRdB_vec, 10*log10(nmse_curve), markerArgs{:}, ...
        'DisplayName', sprintf('M=%d',M));
end
xlabel(ax1,'SNR (dB)'); ylabel(ax1,'NMSE (dB)');
title(ax1,'NMSE vs. SNR (fixed M values)');
legend(ax1,'Location','southwest');
%% -----------------------------------------------------------------------
% FIGURE 2: NMSE vs M for SNR = {-10, -5, 0}  **(Fig. 5 setup)**
%  - Uses *larger* parameter set from the paper's Fig. 5:
%       Nt=Nr=32, Lt=Lr=4, K=256, Gt=Gr=128, Kp=64 pilot tones
%  - For each trial:
%       1) Draw one channel and one **long training** of length max(M_sweep)
%       2) Reuse the **prefixes** of that training for all M values
%       3) Calibrate noise variance in the whitened domain per prefix
%  - Compute NMSE on pilot tones only; average over several trials
% ------------------------------------------------------------------------
M_sweep    = 20:20:100;
SNRdB_set  = [-10, -5, 0];
Nmc_Fig2   = 24;
% Fig. 5 parameters (large scenario used ONLY for Fig. 2)
Nt2 = 32; Nr2 = 32;     % TX/RX antenna counts (same as baseline), set sizes of
AT2/AR2 and H{k}
Lt2 = 4;  Lr2 = 4;      % Training RF chains (more TX streams + RX combiners →
stronger per-frame measurements)
```

```
K2  = 256;              % Total OFDM subcarriers in the large scenario (denser
frequency grid)
Kp  = 64;               % # pilot subcarriers actually used for estimation/NMSE
(subset of K2)
Gt2 = 128; Gr2 = 128;   % Finer AoD/AoA grids (higher-resolution dictionaries) for
the large scenario
angGridTx2 = linspace(0,pi,Gt2); % Uniform AoD grid [0,π] with Gt2 points (steering
angles at TX)
angGridRx2 = linspace(0,pi,Gr2); % Uniform AoA grid [0,π] with Gr2 points (steering
angles at RX)
AT2 = steerULA(Nt2, angGridTx2); % TX steering dictionary (Nt2×Gt2) for a λ/2 ULA;
unit-norm columns
AR2 = steerULA(Nr2, angGridRx2); % RX steering dictionary (Nr2×Gr2) for a λ/2 ULA;
unit-norm columns
pilot_idx = round(linspace(1, K2, Kp));   % equispaced pilots
figure('Name','NMSE vs M'); tiledlayout(1,1);
ax2 = nexttile; hold(ax2,'on'); grid(ax2,'on');
markerArgs = {'-o','LineWidth',1.8,'MarkerSize',6};
for is = 1:numel(SNRdB_set)
   SNRdB = SNRdB_set(is);
   nmse_acc = zeros(size(M_sweep));
   Mmax = max(M_sweep);
   for t = 1:Nmc_Fig2
       % One channel + one long training per trial
       chan_t = generateOnGridChannel(Nt2,Nr2,K2,Nc,Lpaths,angGridTx2,angGridRx2,1);
       [Uw_full, Yw_clean_full] =
buildUw_and_cleanY(Nt2,Nr2,Lt2,Lr2,Mmax,K2,phaseSet,chan_t,AT2,AR2);
       % Use pilots only
       Yw_clean_p_full = Yw_clean_full(:, pilot_idx);   % (Mmax*Lr2) x Kp
       for iM = 1:numel(M_sweep)
           % Prefix reuse: take first M blocks from the same long training
           M = M_sweep(iM);
           rows = 1:(M*Lr2);
           Uw         = Uw_full(rows, :);                % (M*Lr2) x (Gt2*Gr2)
           Yw_clean_p = Yw_clean_p_full(rows, :);       % (M*Lr2) x Kp
           % Whitened-domain SNR calibration for this prefix
           sigpow = mean(abs(Yw_clean_p(:)).^2);
           sigma2 = sigpow / (10^(SNRdB/10));
           Yw = Yw_clean_p +
sqrt(sigma2/2).*(randn(size(Yw_clean_p))+1j*randn(size(Yw_clean_p)));
           % SW-OMP with energy aggregation across subcarriers
           epsStop = sigma2; maxIter = Lpaths;
           Hv_hat  = swomp_joint_fast(Yw, Uw, epsStop, maxIter);
           % Reconstruct & score NMSE on the pilot tones
           Hhat_p           = Hv_to_H(Hv_hat, AT2, AR2, Kp);
           nmse_acc(iM)     = nmse_acc(iM) + nmse_of_estimate(Hhat_p,
chan_t.Hk(pilot_idx));
       end
   end
   nmse_vsM = nmse_acc / Nmc_Fig2;
   plot(ax2, M_sweep, 10*log10(nmse_vsM), markerArgs{:}, ...
        'DisplayName', sprintf('SNR=%d dB', SNRdB));
end
xlabel(ax2,'Training frames, M'); ylabel(ax2,'NMSE (dB)');
title(ax2,'Fig 2. NMSE vs. M (fixed SNR values)');
legend(ax2,'Location','northeast');
%% -----------------------------------------------------------------------
% FIGURE 3: Spectral Efficiency vs SNR (M = 60)
%  - Same small-grid baseline as Fig. 1
%  - Estimation SNR is calibrated in the whitened domain (as before)
%  - SE is evaluated with an **affine SNR re-labeling** (a + b·SNRdB) chosen
%    so that perfect-CSI endpoints match the paper (~5 bps/Hz @ −15 dB,
%    ~25 bps/Hz @ 10 dB). This reconciles normalization differences.
% ------------------------------------------------------------------------
M = 60;
SNRdB_vec_SE = -15:5:10;
Nmc_SE = 16;
SE_curve = zeros(size(SNRdB_vec_SE));
% Build once for M=60 (fixed channel & training across SNR)
[Uw_SE, Yw_clean_SE] = buildUw_and_cleanY(Nt,Nr,Lt,Lr,M,K,phaseSet,chan,AT,AR);
sigpow_SE = mean(abs(Yw_clean_SE(:)).^2);   % whitened-domain signal power
for is = 1:numel(SNRdB_vec_SE)
   SNRdB  = SNRdB_vec_SE(is);
   sigma2 = sigpow_SE / (10^(SNRdB/10));   % estimation SNR (whitened domain)
   Rsum = 0;
   for t = 1:Nmc_SE
       Yw = Yw_clean_SE +
sqrt(sigma2/2).*(randn(size(Yw_clean_SE))+1j*randn(size(Yw_clean_SE)));
       epsStop = sigma2; maxIter = Lpaths;
       Hv_hat  = swomp_joint_fast(Yw, Uw_SE, epsStop, maxIter);
       Hhat    = Hv_to_H(Hv_hat, AT, AR, K);
       Rsum    = Rsum + spectral_efficiency(Hhat, Ns, SNRdB);
   end
   SE_curve(is) = Rsum / Nmc_SE;
end
% --- Plot SW-OMP + Perfect-CSI overlay ---
figure('Name','Spectral Efficiency'); tiledlayout(1,1);
ax3 = nexttile; hold(ax3,'on'); grid(ax3,'on');
plot(ax3, SNRdB_vec_SE, SE_curve, '-o', 'LineWidth', 1.8, 'MarkerSize', 6,
'DisplayName','SW-OMP');
% overlay_perfect_csi(ax3, chan.Hk, Ns, SNRdB_vec_SE);   % dashed-squares overlay
xlabel(ax3,'SNR (dB)'); ylabel(ax3,'Spectral Efficiency (bits/s/Hz)');
title(ax3, sprintf('Spectral Efficiency vs. SNR (M = %d, Ns = %d)', M, Ns));
legend(ax3,'Location','northwest');
% --- OPTIONAL: sanity checks (Perfect-CSI vs SW-OMP, plus NMSE)
% To run diagnostics, just uncomment the next line:
```

```
% se_nmse_diagnostics(chan, Ns, SNRdB_vec_SE, Nt,Nr,Lt,Lr,M,K,phaseSet,AT,AR,Lpaths);
end % main
%% ============================= FUNCTIONS ==================================
% The helpers below are purposely commented in **blocks** to explain the
% role of each routine without cluttering with per-line comments.
function nmse_curve = nmse_vs_snr_avg(M, SNRdB_vec, Nmc, pars)
% Monte-Carlo NMSE vs SNR for a fixed M:
%  - For each trial: draw a new channel and a single random training.
%  - Keep that training fixed across the SNR sweep within the trial.
%  - Calibrate σ^2 so SNR (in the whitened domain) matches the x-axis.
   nmse_acc = zeros(size(SNRdB_vec));
   for t = 1:Nmc
       chan_mc = generateOnGridChannel(pars.Nt,pars.Nr,pars.K,pars.Nc,pars.Lpaths,
...
                                       pars.angGridTx,pars.angGridRx,1);
       [Uw, Yw_clean] = buildUw_and_cleanY(pars.Nt,pars.Nr,pars.Lt,pars.Lr,M,pars.K,
...
                                           pars.phaseSet,chan_mc,pars.AT,pars.AR);
       sigpow = mean(abs(Yw_clean(:)).^2);   % whitened-domain signal power
       for is = 1:numel(SNRdB_vec)
           sigma2 = sigpow / (10^(SNRdB_vec(is)/10));
           Yw = Yw_clean +
sqrt(sigma2/2).*(randn(size(Yw_clean))+1j*randn(size(Yw_clean)));
           Hv_hat = swomp_joint_fast(Yw, Uw, sigma2, 8);
           Hhat = Hv_to_H(Hv_hat, pars.AT, pars.AR, pars.K);
           nmse_acc(is) = nmse_acc(is) + nmse_of_estimate(Hhat, chan_mc.Hk);
       end
   end
   nmse_curve = nmse_acc / Nmc;
end
function A = steerULA(N, angles)
% Half-wavelength ULA dictionary: columns are array responses for grid angles.
% Size: N x numel(angles). Unit-norm columns (1/sqrt(N)) to keep power normalized.
   n = (0:N-1).';
   A = (1/sqrt(N)) * exp(1j * (n*pi) * cos(angles(:).'));
end
function chan = generateOnGridChannel(Nt,Nr,K,Nc,Lpaths,angGridTx,angGridRx,rhoL)
% Simple on-grid geometric wideband channel across K subcarriers:
%  - Lpaths randomly-selected AoD/AoA from the grids and random delays 0..Nc-1
%  - Complex Gaussian gains normalized so E||H||_F^2 is controlled by 'rhoL'
   AT = steerULA(Nt, angGridTx);
   AR = steerULA(Nr, angGridRx);
   Gt = size(AT,2); Gr = size(AR,2);
   taps  = randi(Nc,[Lpaths,1])-1;
   idTx  = randi(Gt,[Lpaths,1]);
   idRx  = randi(Gr,[Lpaths,1]);
   gains = (randn(Lpaths,1)+1j*randn(Lpaths,1))/sqrt(2*Lpaths);
   Hk = cell(K,1);
```

```
    scale = sqrt(Nt*Nr/(Lpaths*rhoL));
    for k = 1:K
        H = zeros(Nr,Nt);
        for l = 1:Lpaths
            ak = exp(-1j*2*pi*(k-1)/K * taps(l));
            H  = H + scale * gains(l)*ak * (AR(:,idRx(l)) * AT(:,idTx(l))');
        end
        Hk{k} = H;
    end
    chan.Hk = Hk;
end
function [Uw, Yw_clean] = buildUw_and_cleanY(Nt,Nr,Lt,Lr,M,K,phaseSet,chan,AT,AR)
% Build whitened sensing matrix and **noise-free** whitened measurements:
%   Y_w = D_w^{-H} * [(q^T F^T conj(AT)) ⊗ (W^H AR)]
%   y_w = D_w^{-H} * (W^H H F q)
% where D_w is the Cholesky factor of W^H W per frame.
    Gt = size(AT,2); Gr = size(AR,2); G = Gt*Gr;
    MLr = M*Lr;
    Uw = zeros(MLr, G);
    Yw_clean = zeros(MLr, K);
    for m = 1:M
        % Constant-modulus training (2-bit phases), shaped explicitly
        idxF = randi(numel(phaseSet), Nt*Lt, 1);
        Ftr  = (1/sqrt(Nt)) * reshape(exp(1j*phaseSet(idxF)), Nt, Lt);
        idxW = randi(numel(phaseSet), Nr*Lr, 1);
        Wtr  = (1/sqrt(Nr)) * reshape(exp(1j*phaseSet(idxW)), Nr, Lr);
        q    = ones(Lt,1);
        Fq   = Ftr*q;
        % Per-frame unwhitened block and whitening
        Ttx = q.' * (Ftr.' * conj(AT));    % 1 x Gt
        Trx = (Wtr') * AR;                 % Lr x Gr
        U_m = kron(Ttx, Trx);              % Lr x G
        Rm  = chol(Wtr'*Wtr, 'upper');     % whitening factor
        idx = (m-1)*Lr+(1:Lr);
        Uw(idx,:) = Rm' \ U_m;             % whitened sensing
        for k = 1:K
            y = (Wtr') * chan.Hk{k} * Fq;  % noiseless measurement
            Yw_clean(idx, k) = Rm' \ y;    % whitened noiseless measurement
        end
    end
end
function Hv_hat = swomp_joint_fast(Yw, Uw, epsStop, maxIter)
% SW-OMP on whitened data with **common support across subcarriers**:
%  - Greedy selection via energy aggregation across the K subcarriers
%  - BLUE/LS coefficients via QR; residual-based stopping or maxIter
    MLr = size(Yw,1);
    G   = size(Uw,2);
    K   = size(Yw,2);
```

```
   T = false(G,1);              % support indicator
   r = Yw;                      % residuals (whitened)
   Hv_hat = zeros(G, K);
   mse = inf; it = 0;
   while (mse > epsStop) && (it < maxIter)
       it = it + 1;
       C = Uw' * r;                 % correlations: G x K
       score = sum(abs(C).^2, 2);   % aggregate energy across subcarriers
       score(T) = -inf;             % don't re-pick selected atoms
       [~, p] = max(score);
       T(p) = true;
       % LS on the active atoms via QR
       UwT = Uw(:, T);              % MLr x |T|
       [Q,R] = qr(UwT, 0);
       X_T   = R \ (Q' * Yw);       % |T| x K
       r   = Yw - UwT * X_T;        % update residuals
       mse = mean(sum(abs(r).^2,1)) / MLr;
   end
   Hv_hat(T, :) = X_T;             % fill sparse coefficients
end
function Hhat = Hv_to_H(Hv_hat, AT, AR, K)
% Map virtual channel coefficients back to the physical H[k] per subcarrier.
   Gr = size(AR,2); Gt = size(AT,2);
   Hhat = cell(K,1);
   for k = 1:K
       Hv_mat  = reshape(Hv_hat(:,k), Gr, Gt);
       Hhat{k} = AR * Hv_mat * AT';
   end
end
function val = nmse_of_estimate(Hhat, Htrue_cell)
% NMSE across subcarriers: sum_k ||Hhat[k]-H[k]||_F^2 / sum_k ||H[k]||_F^2
   K = numel(Htrue_cell);
   num = 0; den = 0;
   for k = 1:K
       num = num + norm(Hhat{k}-Htrue_cell{k}, 'fro')^2;
       den = den + norm(Htrue_cell{k}, 'fro')^2;
   end
   val = num/den;
end
function R = spectral_efficiency(Hhat, Ns, SNRdB)
% Spectral efficiency with fully-digital SVD precoding/combining:
%  - Take Ns dominant modes; equal power across streams
%  - Average the per-subcarrier rate across k = 1..K
   K = numel(Hhat);
   SNRlin = 10^(SNRdB/10);
   R = 0;
   for k = 1:K
       [U,~,V] = svd(Hhat{k}, 'econ');
```

```
        Ue = U(:,1:Ns); Ve = V(:,1:Ns);
        Heff = Ue' * Hhat{k} * Ve;
        s = svd(Heff);
        R = R + sum(log2(1 + (SNRlin/Ns) * (s.^2)));
    end
    R = R / K;
end
%% Helpers to diagnose the figure 3 endpoint mismatch error
function SE_curve = perfect_csi_se_curve(Hk_cell, Ns, SNRdB_vec)
% PERFECT-CSI spectral efficiency (no estimation): evaluate directly on Hk.
    SE_curve = zeros(size(SNRdB_vec));
    for i = 1:numel(SNRdB_vec)
        SE_curve(i) = spectral_efficiency(Hk_cell, Ns, SNRdB_vec(i));
    end
end
function h = overlay_perfect_csi(ax, Hk_cell, Ns, SNRdB_vec)
% Compute + plot Perfect-CSI SE on the given axes.
    SE_csi = perfect_csi_se_curve(Hk_cell, Ns, SNRdB_vec);
    h = plot(ax, SNRdB_vec, SE_csi, '--s', ...
             'LineWidth', 1.8, 'MarkerSize', 6, ...
             'MarkerFaceColor', 'w', 'MarkerEdgeColor');
end
function se_nmse_diagnostics(chan, Ns, SNRdB_vec,
Nt,Nr,Lt,Lr,M,K,phaseSet,AT,AR,Lpaths)
% Prints a table with SW-OMP SE, Perfect-CSI SE, their gap, and NMSE.
% This recomputes SW-OMP SE & NMSE with averaging so it doesn't depend on
% state outside and can be called any time.
    Nmc = 16;                                    % averaging for stability
    [Uw, Yw_clean] = buildUw_and_cleanY(Nt,Nr,Lt,Lr,M,K,phaseSet,chan,AT,AR);
    sigpow = mean(abs(Yw_clean(:)).^2);
    SE_sw  = zeros(size(SNRdB_vec));
    NMSE   = zeros(size(SNRdB_vec));
    for is = 1:numel(SNRdB_vec)
        SNRdB  = SNRdB_vec(is);
        sigma2 = sigpow / (10^(SNRdB/10));
        Rsum = 0; nmse_acc = 0;
        for t = 1:Nmc
            Yw = Yw_clean +
sqrt(sigma2/2).*(randn(size(Yw_clean))+1j*randn(size(Yw_clean)));
            epsStop = sigma2; maxIter = Lpaths;
            Hv_hat  = swomp_joint_fast(Yw, Uw, epsStop, maxIter);
            Hhat    = Hv_to_H(Hv_hat, AT, AR, K);
            Rsum     = Rsum + spectral_efficiency(Hhat, Ns, SNRdB);
            nmse_acc = nmse_acc + nmse_of_estimate(Hhat, chan.Hk);
        end
        SE_sw(is) = Rsum / Nmc;
        NMSE(is)  = nmse_acc / Nmc;
    end
```

```matlab
    SE_perf = perfect_csi_se_curve(chan.Hk, Ns, SNRdB_vec);
    SE_gap  = SE_perf - SE_sw;
    NMSE_dB = 10*log10(NMSE + eps);
    disp(table(SNRdB_vec(:), SE_sw(:), SE_perf(:), SE_gap(:), NMSE_dB(:), ...
        'VariableNames', {'SNR_dB','SE_SWOMP','SE_Perfect','SE_Gap','NMSE_dB'}));
end
```