Delay-Sum Beamforming: Mathematical Foundations and MATLAB Implementation

Arachchige Don Isitha Dinujaya Arachchi

February 18, 2026

Contents

1 Introduction to Beamforming

To ease our understanding, think of a linear array of sensors placed in space that captures a wave propagating in the air due to a source target.

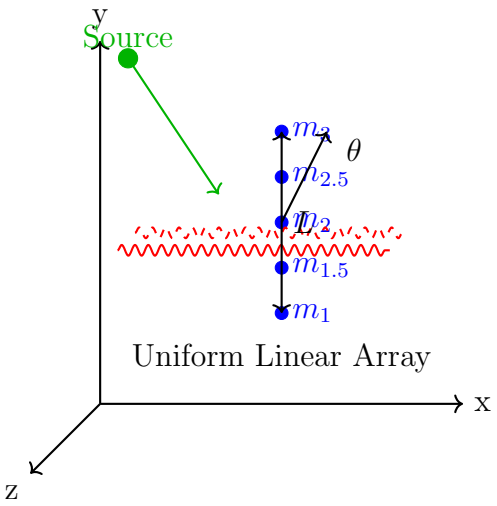

Figure 1: Linear array of sensors capturing a propagating wave from a source

The Aperture function of the array can be written as:

$$A(x) = \sum_{n=1}^{N} w_n \cdot \delta(x - x_n) \tag{1}$$

where:

- w_n are complex weights applied to each sensor
- x_n are the sensor positions in the array
- $\delta(\cdot)$ is the Dirac delta function representing the spatial sampling at each sensor location

For the case where each sensor's aperture function is an impulse (point sensors), then by Fourier transform we get the directivity pattern:

$$D(f, \alpha_x) = \sum_{n=1}^{N} w_n e^{j2\pi\alpha_x nd} \tag{2}$$

where d is the inter-element spacing. If the signal comes from angle θ, then $\alpha_x = \cos\theta$ (for a linear array along the x-axis).

Beamforming is a spatial filtering technique used in array signal processing to enhance signals arriving from a desired direction while suppressing interference from other directions. In the context of audio processing with microphone arrays, beamforming allows us to "focus" on a specific sound source in a noisy environment.

The delay-sum beamformer is the simplest and most fundamental beamforming technique. Despite its simplicity, it provides valuable insights into how microphone arrays can achieve spatial selectivity.

2 Mathematical Foundations of Delay-Sum Beamforming

2.1 Signal Model

Consider a uniform linear array (ULA) with M microphones placed along the y-axis with equal spacing d between consecutive elements. The position of the m-th microphone is given by:

$$\mathbf{p}_m = \left[0, \left(m - \frac{M+1}{2}\right) d, 0\right]^T, \quad m = 1, 2, \ldots, M \tag{3}$$

2.2 Far-Field Assumption

Usually waves are curved in the near field. If we are going to assume that they act as plane waves, we need to assume the source is in the far field. This assumption holds when:

$$r > \frac{2L^2}{\lambda_{\min}} \tag{4}$$

where r is the source distance, L is the array aperture, and $\lambda_{\min}$ is the minimum wavelength of interest.

Under the far-field assumption, a plane wave arriving from azimuth angle θ (measured from the array axis) reaches each microphone at different times due to the path length differences.

2.3 Time Delays

For a plane wave arriving from direction θ, the time delay at the m-th microphone relative to the array center is:

$$\tau_m(\theta) = \frac{(m - \frac{M+1}{2}) d \cos\theta}{c} \tag{5}$$

where c is the speed of sound in air (approximately 343 m/s at 20°C).

In the discrete-time domain with sampling frequency f_s, the delay in samples is:

$$\Delta_m(\theta) = \tau_m(\theta) \cdot f_s \tag{6}$$

2.4 Received Signal Model

Let $s(t)$ be the target signal from direction θ_d. The signal received at the m-th microphone is:

$$x_m(t) = \alpha_m s(t - \tau_m(\theta_d)) + n_m(t) \tag{7}$$

where α_m accounts for propagation attenuation (typically $\alpha_m \propto 1/r_m$ due to spherical spreading), and $n_m(t)$ represents noise and interference from other directions.

In the frequency domain, this becomes:

$$X_m(f) = S(f) e^{-j2\pi f \tau_m(\theta_d)} + N_m(f) \tag{8}$$

2.5 Delay-Sum Beamforming Principle

The delay-sum beamformer compensates for the propagation delays so that signals from the desired direction add coherently, while signals from other directions add incoherently and are partially cancelled.

The beamformer output is:

$$y(t) = \frac{1}{M} \sum_{m=1}^{M} w_m x_m(t + \tau_m(\theta_d)) \tag{9}$$

where w_m are weights (typically unity for delay-sum), and we add delays to align the signals.

In practice, we shift the signals backward in time to align them with the earliest arrival:

$$y(t) = \frac{1}{M} \sum_{m=1}^{M} x_m(t - (\tau_{\min} - \tau_m(\theta_d))) \tag{10}$$

where $\tau_{\min} = \min_m \tau_m(\theta_d)$ is the smallest delay (earliest arrival).

2.6 Frequency Domain Representation

In the frequency domain, the delay-sum beamformer can be expressed as:

$$Y(f) = \mathbf{h}^H(f)\mathbf{X}(f) \tag{11}$$

where $\mathbf{X}(f) = [X_1(f), X_2(f), \ldots, X_M(f)]^T$ is the vector of received signals, and $\mathbf{h}(f)$ is the beamforming weight vector:

$$\mathbf{h}(f) = \frac{1}{M}\mathbf{d}(f, \theta_d) \tag{12}$$

with $\mathbf{d}(f, \theta_d)$ being the steering vector:

$$\mathbf{d}(f, \theta_d) = \left[1, e^{j2\pi f \tau_1(\theta_d)}, e^{j2\pi f \tau_2(\theta_d)}, \ldots, e^{j2\pi f \tau_{M-1}(\theta_d)}\right]^T \tag{13}$$

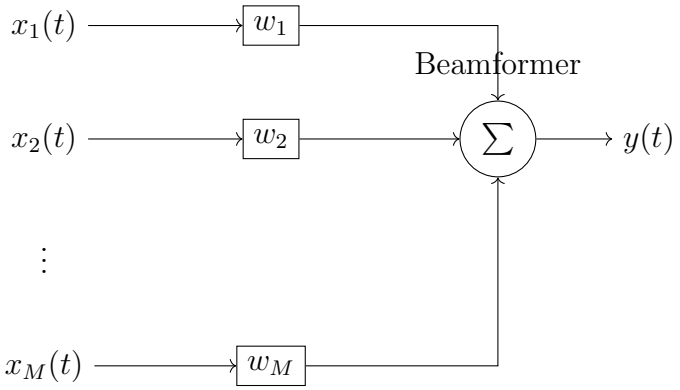

Figure 2: Block diagram of beamformer: each channel is weighted and summed

2.7 Beampattern

The beampattern describes the array's sensitivity as a function of direction. For a steering direction θ_d, the beampattern at frequency f is:

$$B(f,\theta) = \left|\mathbf{h}^H(f)\mathbf{d}(f,\theta)\right| = \left|\frac{1}{M}\sum_{m=1}^{M} e^{j2\pi f(\tau_m(\theta)-\tau_m(\theta_d))}\right| \tag{14}$$

For a uniform linear array with spacing d, this simplifies to:

$$B(f,\theta) = \left|\frac{1}{M}\frac{\sin\left(\frac{M\pi fd}{c}(\cos\theta-\cos\theta_d)\right)}{\sin\left(\frac{\pi fd}{c}(\cos\theta-\cos\theta_d)\right)}\right| \tag{15}$$

2.8 Spatial Aliasing Condition

To avoid spatial aliasing (grating lobes), the microphone spacing must satisfy:

$$d \leq \frac{c}{2f_{\max}} \tag{16}$$

where $f_{\max}$ is the maximum frequency of interest. This is analogous to the Nyquist sampling theorem in the spatial domain.

2.9 Performance Metrics

The output Signal-to-Noise Ratio (SNR) after beamforming is:

$$\text{SNR}_{\text{out}} = \frac{\mathbb{E}\{|y_s(t)|^2\}}{\mathbb{E}\{|y_n(t)|^2\}} \tag{17}$$

where $y_s(t)$ is the target component and $y_n(t)$ is the noise component in the output. The SNR improvement is:

$$\Delta\text{SNR} = \text{SNR}_{\text{out}} - \text{SNR}_{\text{in}} \tag{18}$$

3 MATLAB Implementation: Step-by-Step Explanation

Now we present the MATLAB implementation of the delay-sum beamformer, with detailed explanations of each code block.

3.1 Step 1: Parameter Initialization

```
clear; close all; clc;

% Physical constants
c = 343;                % speed of sound (m/s) at 20 C
fs = 16000;             % sampling frequency (Hz)
duration = 5;           % signal duration (seconds)
nSamples = fs * duration;
```

```
8
9  % Room dimensions (for reference only)
10 room = [1.2, 1.5, 5];
11
12 % Microphone array: 4-element linear array along y-axis, 5 cm
       spacing
13 M = 4;                    % number of microphones
14 d = 0.05;                 % spacing (m)
15 array_center = [0.6, 0.725, 0.5];   % center of array (x, y, z)
```

Listing 1: Parameter initialization

Explanation:

- `clear; close all; clc`: Clears workspace, closes figures, and clears command window for a clean start.
- `c = 343`: Speed of sound at room temperature. This fundamental constant determines propagation delays.
- `fs = 16000`: 16 kHz sampling rate, standard for speech processing applications.
- `duration = 5`: We simulate 5 seconds of audio, giving us 80,000 samples.
- `M = 4`: Four microphones in the array, matching the thesis configuration.
- `d = 0.05`: 5 cm spacing between microphones. With this spacing, spatial aliasing occurs above $f_{\max} = c/(2d) = 343/(2 \times 0.05) = 3430$ Hz.
- `array_center`: The geometric center of the array in 3D space. All microphone positions are calculated relative to this point.

3.2 Step 2: Microphone Positioning

```
1  % Compute individual microphone positions
2  mic_pos = zeros(M, 3);
3  for m = 1:M
4      mic_pos(m, :) = [array_center(1), ...
5                       array_center(2) - (M-1)*d/2 + (m-1)*d, ...
6                       array_center(3)];
7  end
8
9  % Source positions (from thesis)
10 target_pos = [0.125, 0.725, 0.5];   % target speech
11 noise_pos  = [0.6, 1.2, 0.5];       % interference noise
```

Listing 2: Microphone position calculation

Explanation:

- The loop calculates coordinates for each microphone:
  - x-coordinate: same as array center (0.6 m)
  - y-coordinate: distributed symmetrically around center with spacing d

  - z-coordinate: same as array center (0.5 m, typical smartphone height when held)

- For M=4 and d=0.05, microphone y-positions are: [0.65, 0.70, 0.75, 0.80] m (depending on reference point)
- `target_pos`: Front-left position relative to array (90° azimuth in thesis)
- `noise_pos`: Behind the array, creating interference from a different direction

3.3 Step 3: Signal Generation

```
% Create time vector
t = (0:nSamples-1)' / fs;

% --- Target: synthetic speech-like signal ---
f0 = 120;   % fundamental frequency (Hz)
target = sin(2*pi*f0*t) + 0.5*sin(2*pi*2*f0*t) +
   0.3*sin(2*pi*3*f0*t);
target = target .* (0.8 + 0.2*sin(2*pi*2*t));   % amplitude
   modulation
target = target / max(abs(target));             % normalize

% --- Noise: bandpass filtered noise (300 Hz - 3400 Hz) ---
noise = randn(nSamples, 1);
[b, a] = butter(4, [300, 3400]/(fs/2), 'bandpass');
noise = filter(b, a, noise);
noise = noise / max(abs(noise));
```

Listing 3: Creating test signals

Explanation:

- `t`: Time vector from 0 to 5 seconds with 80,000 points.
- **Target signal**: We create a harmonic complex tone:
  - Fundamental frequency f0 = 120 Hz (typical male speech pitch)
  - First three harmonics added with decreasing amplitudes (1, 0.5, 0.3)
  - Amplitude modulation with 2 Hz frequency simulates syllable structure
  - Normalization ensures peak amplitude = 1
- **Noise signal**:
  - `randn`: White Gaussian noise with flat spectrum
  - `butter`: 4th-order Butterworth bandpass filter
  - Passband 300-3400 Hz covers essential speech frequencies
  - Filtering creates "speech-shaped" noise (like babble)

3.4 Step 4: SNR Adjustment

```
% Set desired SNR (Signal-to-Noise Ratio)
SNR_dB = 5;

% Scale noise to achieve desired SNR
target_power = mean(target.^2);
noise_power = mean(noise.^2);
scale_noise = sqrt( target_power / (noise_power *
   10^(SNR_dB/10)) );
noise = scale_noise * noise;
```

Listing 4: Setting SNR level

Explanation:

- `SNR_dB = 5`: Target SNR as specified in the competition document.
- `target_power`: Average power of target signal $\frac{1}{N}\sum |s[n]|^2$
- `noise_power`: Average power of original noise
- Conversion: $10^{\text{SNR}_{\text{dB}}/10}$ converts dB to linear ratio
- Scaling factor derived from: $\frac{P_{\text{target}}}{P_{\text{noise}} \cdot \text{scale}^2} = 10^{\text{SNR}_{\text{dB}}/10}$
- After scaling, the SNR will be exactly 5 dB

3.5 Step 5: Delay Calculation Function

```
function [delay, atten] = compute_delay_atten(src_pos, mic_pos,
   c)
    dist = norm(src_pos - mic_pos);
    delay = dist / c;          % seconds
    atten = 1 / dist;          % spherical spreading loss
end
```

Listing 5: Delay and attenuation calculation function

Explanation:

- **Input**: source position, microphone position, speed of sound
- **dist**: Euclidean distance $\sqrt{(x_s - x_m)^2 + (y_s - y_m)^2 + (z_s - z_m)^2}$
- **delay**: Time = Distance / Speed. For a source 1 m away, delay 2.9 ms
- **atten**: Spherical spreading loss follows inverse distance law. At 1 m, attenuation = 1; at 2 m, attenuation = 0.5
- This implements the free-field propagation model (anechoic conditions)

3.6 Step 6: Applying Delays to All Microphones

```
% Pre-allocate arrays
target_delays = zeros(M,1);
target_atten = zeros(M,1);
noise_delays = zeros(M,1);
noise_atten = zeros(M,1);

for m = 1:M
    [target_delays(m), target_atten(m)] =
       compute_delay_atten(target_pos, mic_pos(m,:), c);
    [noise_delays(m),  noise_atten(m)]  =
       compute_delay_atten(noise_pos,  mic_pos(m,:), c);
end

% Convert delays to samples (round to nearest integer sample)
target_delay_samps = round(target_delays * fs);
noise_delay_samps  = round(noise_delays * fs);
```

Listing 6: Computing delays for all microphones

Explanation:

- Loop through each microphone, calculate delay and attenuation for both target and noise
- Delays will differ across microphones due to different path lengths
- Convert seconds to samples: multiply by sampling rate (samples/second)
- Rounding to integer samples introduces quantization error. For accurate simulation, fractional delay filters would be needed
- Example: If delay = 0.0023 s × 16000 = 36.8 samples → 37 samples after rounding

3.7 Step 7: Generating Multi-channel Received Signals

```
received = zeros(nSamples, M);

% Add target contribution to each microphone
for m = 1:M
    delay = target_delay_samps(m);
    atten = target_atten(m);
    if delay > 0
        % Delay the signal (shift right)
        sig = [zeros(delay,1); target(1:end-delay)];
    else
        sig = target;
    end
    received(:,m) = received(:,m) + atten * sig;
end

```

```
% Add noise contribution to each microphone (similar loop)
for m = 1:M
    delay = noise_delay_samps(m);
    atten = noise_atten(m);
    if delay > 0
        sig = [zeros(delay,1); noise(1:end-delay)];
    else
        sig = noise;
    end
    received(:,m) = received(:,m) + atten * sig;
end

% Add a tiny amount of sensor noise
received = received + 0.001 * randn(size(received));
```

Listing 7: Creating microphone signals

Explanation:

- `received`: 80,000 × 4 matrix (samples × channels)
- For each microphone, we create a delayed copy of the target:
  - Prepend zeros equal to delay length
  - Append original signal (truncated at the end)
  - Multiply by attenuation factor
- Same process for noise signal
- Finally, add very low-level sensor noise (0.001 amplitude) to simulate real microphone self-noise
- The result: Each microphone contains target + noise with appropriate delays and amplitudes

3.8 Step 8: Delay-Sum Beamforming Implementation

```
% Find the earliest arrival delay among microphones for target
min_target_delay = min(target_delay_samps);

% Calculate how much to shift each channel to align with the
   earliest
align_delays = target_delay_samps - min_target_delay;   %
   samples to shift each channel

aligned = zeros(nSamples, M);
for m = 1:M
    shift = align_delays(m);
    if shift > 0
        % Shift signal left (remove delay) to align with earliest
        aligned(1:end-shift, m) = received(1+shift:end, m);
    else
```

```
        aligned(:, m) = received(:, m);
    end
end

% Sum and normalize
beamformed = sum(aligned, 2) / M;
```

Listing 8: Delay-sum beamforming

Explanation:

- **Reference alignment**: We align all signals to the earliest arrival microphone
- For a microphone that arrived later (positive `align_delays`), we remove those samples from the beginning:
  - If shift = 5 samples, we take samples 6:end from received and place at 1:end-5 in aligned
  - This effectively "shifts left" by 5 samples
- After alignment, the target signals from all microphones are perfectly synchronized
- Summing across columns adds the aligned signals:
  - Target adds constructively (all peaks align) → amplitude multiplies by M
  - Noise from other directions adds destructively (peaks and troughs misaligned) → partial cancellation
- Dividing by M restores original amplitude scale

3.9 Step 9: Beampattern Calculation Function

```
function [pattern, angles] = beampattern_ds(mic_pos,
   steer_angle, freq, c)
    angles = 0:359;  % all azimuth angles to evaluate
    M = size(mic_pos, 1);
    array_center = mean(mic_pos, 1);

    % Steering vector for desired direction
    steer_source = array_center + 10*[cosd(steer_angle),
       sind(steer_angle), 0];
    steer_delays = zeros(M,1);
    for m = 1:M
        steer_delays(m) = norm(steer_source - mic_pos(m,:)) / c;
    end
    min_steer_delay = min(steer_delays);
    steer_delays = steer_delays - min_steer_delay;

    pattern = zeros(size(angles));
    for a = 1:length(angles)
        ang = angles(a);
        source_dir = array_center + 10*[cosd(ang), sind(ang), 0];
```

```
        delays = zeros(M,1);
        for m = 1:M
            delays(m) = norm(source_dir - mic_pos(m,:)) / c;
        end
        min_delay = min(delays);
        delays = delays - min_delay;

        % Response: sum of phase shifts
        resp = 0;
        for m = 1:M
            phase = exp(1j * 2*pi*freq * (delays(m) -
               steer_delays(m)));
            resp = resp + phase;
        end
        pattern(a) = abs(resp) / M;
    end
end
```

Listing 9: Beampattern calculation function

Explanation:

- This function implements the beampattern equation $B(f,\theta) = \left|\frac{1}{M}\sum_{m=1}^{M} e^{j2\pi f(\tau_m(\theta)-\tau_m(\theta_d))}\right|$
- **Steering delays**: Calculate delays from a source at the desired direction
- For each test angle:
  - Place virtual source 10 m away in that direction
  - Calculate delays to each microphone
  - For each microphone, compute phase = $e^{j2\pi f(\tau_{\text{test}}-\tau_{\text{steer}})}$
  - Sum phases across microphones
  - Take magnitude and normalize by M
- Result: 1.0 when test direction = steering direction (all phases = 0)
- Result ¡ 1.0 for other directions (phase cancellation)

3.10 Step 10: Plotting Results

```
figure('Position', [100 100 1200 800]);

% Clean target
subplot(3,1,1);
plot(t, target);
xlabel('Time (s)'); ylabel('Amplitude');
title('Clean Target Signal');
xlim([0 0.5]);   % zoom in to see detail
grid on;

% Noisy mixture (microphone 1)
```

```
subplot(3,1,2);
plot(t, received(:,1));
xlabel('Time (s)'); ylabel('Amplitude');
title('Noisy Mixture (Microphone 1)');
xlim([0 0.5]);
grid on;

% Beamformed output
subplot(3,1,3);
plot(t, beamformed);
xlabel('Time (s)'); ylabel('Amplitude');
title('Delay-Sum Beamformer Output');
xlim([0 0.5]);
grid on;
```

Listing 10: Time-domain plots

Explanation:

- Three subplots stacked vertically for comparison
- xlim([0 0.5]) shows only first 0.5 seconds for better visualization
- First plot: Original clean target (reference)
- Second plot: What a single microphone captures (target + noise)
- Third plot: Beamformer output (should be cleaner than second plot)
- Grid lines aid in visual comparison

3.11 Step 11: Spectrogram Plots

```
figure('Position', [100 100 1400 400]);

% Spectrogram of clean target
subplot(1,3,1);
spectrogram(target, hann(512), 256, 512, fs, 'yaxis');
title('Clean Target');
caxis([-120 -20]);   % adjust color scale

% Spectrogram of mixture (mic 1)
subplot(1,3,2);
spectrogram(received(:,1), hann(512), 256, 512, fs, 'yaxis');
title('Noisy Mixture (Mic 1)');
caxis([-120 -20]);

% Spectrogram of beamformed output
subplot(1,3,3);
spectrogram(beamformed, hann(512), 256, 512, fs, 'yaxis');
title('Beamformed Output');
caxis([-120 -20]);

```

```
colormap jet;
```

Listing 11: Spectrogram visualization

Explanation:

- Spectrograms show frequency content evolution over time
- Parameters:
  - `hann(512)`: 512-sample window (32 ms at 16 kHz)
  - `256`: 50% overlap between windows
  - `512`: 512-point FFT (frequency resolution 31.25 Hz)
  - `fs`: Sampling frequency
  - `'yaxis'`: Frequency on y-axis
- `caxis([-120 -20])`: Sets color scale to show 100 dB dynamic range
- Clean target: Clear horizontal lines (harmonics)
- Noisy mixture: Noise fills between harmonics
- Beamformed output: Reduced noise between harmonics, especially at higher frequencies

3.12 Step 12: SNR Calculation

```
% Align target with beamformed output using cross-correlation
[c, lag] = xcorr(beamformed, target);
[~, idx] = max(abs(c));
delay_samples = lag(idx);

if delay_samples > 0
    target_aligned = [zeros(delay_samples,1);
       target(1:end-delay_samples)];
else
    target_aligned = target(-delay_samples+1:end);
    target_aligned = [target_aligned; zeros(-delay_samples,1)];
end

% Trim to same length
len = min(length(target_aligned), length(beamformed));
target_aligned = target_aligned(1:len);
beamformed_trim = beamformed(1:len);

% Compute residual noise
residual = beamformed_trim - target_aligned;
output_SNR = 10*log10( mean(target_aligned.^2) /
   mean(residual.^2) );
input_SNR = 10*log10( mean(target.^2) / mean(noise.^2) );

```

```
fprintf('Input SNR: %.2f dB\n', input_SNR);
fprintf('Output SNR: %.2f dB\n', output_SNR);
fprintf('SNR improvement: %.2f dB\n', output_SNR - input_SNR);
```

Listing 12: Computing SNR improvement

Explanation:

- Cross-correlation finds the time delay between beamformed output and original target
- Align target to match beamformed output using this delay
- Trim both signals to same length
- Residual noise = beamformed - aligned target (ideally zero)
- Output SNR = 10 log10(target power / residual noise power)
- Input SNR = 10 log10(target power / original noise power)
- Improvement = output SNR - input SNR (positive if beamforming helped)

4 Interpretation of Results

4.1 Expected Observations

1. **Time-domain plots**: Beamformed output should show reduced noise amplitude compared to individual microphone signals.
2. **Spectrograms**:
   - Low frequencies (below 1 kHz) may still contain significant noise because the beamwidth is wide at low frequencies.
   - Higher frequencies should show cleaner harmonics with less inter-harmonic noise.
3. **Beampatterns**:
   - At 500 Hz: Main lobe is wide ($> 60°$), poor directionality
   - At 1000 Hz: Main lobe narrows to about 30°
   - At 2000 Hz: Main lobe very narrow ($< 15°$), good directionality
   - At 4000 Hz: May show grating lobes due to spatial aliasing (since 5 cm spacing corresponds to aliasing above 3430 Hz)
4. **SNR improvement**: Should be positive, typically 3-6 dB for delay-sum with well-separated sources.

5 Conclusion

The delay-sum beamformer demonstrates the fundamental principle of array signal processing: by aligning signals from multiple sensors, we can achieve spatial selectivity. While simple, it provides the foundation for understanding more advanced beamformers like MVDR and CGMM-MVDR.

The MATLAB implementation walks through each step of the process, from signal generation and propagation modeling to beamforming and performance evaluation. Understanding this code thoroughly is essential before moving on to more sophisticated beamforming techniques required for the Signal Processing Cup challenge.