Let's break down the entire concept step by step, starting from the basics and gradually moving to the code. This explanation assumes no prior knowledge of the topic, aiming to make it accessible to a high school student.

**1. What Are Antennas and Their Purpose?**

Antennas are devices used to transmit and receive signals, like the ones in your phone, Wi-Fi router, or TV. A signal is just information, like music, data, or video, sent as electromagnetic waves through the air.

**2. What Is an Antenna Array?**

Instead of using one antenna, we use a group of antennas arranged in a pattern. This group is called an *antenna array*. By arranging antennas in specific ways, we can make the signal stronger in certain directions, which helps in better communication.

* Think of it like multiple microphones in a room: by combining their outputs, you can "focus" on the sound coming from a particular direction.
* In this project, we use a **rectangular array** (antennas arranged in rows and columns).

**3. What Is a Metasurface?**

A *metasurface* is a special surface made of small elements (like antennas) that can control how electromagnetic waves behave. These surfaces can change the direction or strength of the signals passing through them.

**4. How Does the Antenna Array Work?**

Each antenna in the array emits a signal. When the signals combine, the overall strength (or "intensity") in a particular direction depends on:

1. **Position of each antenna in the array.**
2. **Angles from which the signal comes or goes (called theta and phi).**

This process of combining signals to focus in specific directions is called **beamforming**.

**5. What Is a Sinusoidal Signal?**

A sinusoidal wave looks like smooth waves you might see in a swimming pool. It is the simplest form of a signal, like a single musical note. In the code:

* We create a simple sinusoidal signal at 100 Hz (100 waves per second).
* The formula for this wave is: signal(t)=A⋅cos⁡(2πft)\text{signal}(t) = A \cdot \cos(2 \pi f t)signal(t)=A⋅cos(2πft) where:
  + AAA is the amplitude (how "tall" the wave is).
  + fff is the frequency (how many waves per second).
  + ttt is time.

**6. What Is Noise?**

When signals travel, they often pick up unwanted "extra" signals called **noise**, which makes the received signal harder to interpret. Imagine static noise when tuning a radio. We simulate this noise to test how well the system works in real-world conditions.

**7. What Is the Goal of the Code?**

The code simulates how an array of antennas on a metasurface receives signals from different angles and with varying noise levels. The final goal is to:

1. **Calculate the signal received by the array.**
2. **Understand how the direction of arrival (theta and phi) affects the received signal.**

**8. Understanding the Code**

Now, let’s break the code step by step:

**8.1. Parameters**

matlab

Copy code

Nx = 4; % Number of antennas in the x-direction

Ny = 4; % Number of antennas in the y-direction

d = 0.5; % Distance between antennas (in wavelengths)

lambda = 1; % Wavelength of the signal

k = 2 \* pi / lambda; % Wave number (related to the wavelength)

* **Nx, Ny**: Define the grid of antennas (4x4 in this case).
* **d**: How far apart the antennas are (half a wavelength in this case).
* **lambda**: The wavelength of the signal.
* **k**: A constant that helps calculate the phase of the signal.

**8.2. Generate the Signal**

matlab

Copy code

fs = 1e3; % Sampling frequency (how many samples per second)

t = 0:1/fs:1; % Time vector (from 0 to 1 second)

A = 1; % Amplitude of the signal

signal = A \* cos(2 \* pi \* 100 \* t); % A sinusoidal wave at 100 Hz

* The signal is generated as a simple wave with frequency 100 Hz100 \, \text{Hz}100Hz.

**8.3. Antenna Positions**

matlab

Copy code

[x, y] = meshgrid(0:Nx-1, 0:Ny-1); % Create a grid of antenna positions

x = x \* d; % Scale positions in x-direction

y = y \* d; % Scale positions in y-direction

* The antennas are arranged in a 4x4 grid, with each spaced 0.5 λ0.5 \, \lambda0.5λ apart.

**8.4. Loop Through Signal Conditions**

matlab

Copy code

for SNR = -5:2:5 % Loop over signal-to-noise ratios

for theta = -90:5:90 % Loop over incoming signal angles (theta)

for phi = -90:5:90 % Loop over incoming signal angles (phi)

% Calculate and store received signal

end

end

end

* **SNR**: Measures how strong the signal is compared to noise.
* **Theta and Phi**: Angles describing the direction from which the signal arrives.

**8.5. Array Factor**

matlab

Copy code

function Array\_factor = ArrayFactor(theta, phi, Nx, Ny, x, y, k)

Array\_factor = 0;

for i = 1:Nx

for j = 1:Ny

phase\_shift = k \* (x(i,j) \* sin(deg2rad(theta)) \* cos(deg2rad(phi)) + ...

y(i,j) \* sin(deg2rad(theta)) \* sin(deg2rad(phi)));

Array\_factor = Array\_factor + exp(1j \* phase\_shift);

end

end

end

* The **Array Factor** calculates how the antennas combine their signals for a given direction (θ,ϕ\theta, \phiθ,ϕ).
* It uses the positions of antennas and the angles to compute a "phase shift" (how delayed each signal is).

**8.6. Add Noise**

matlab

Copy code

noise = (1 / 10^(SNR / 10)) \* randn(size(received\_signal));

signal\_out = received\_signal + noise;

* Random noise is added to the signal based on the **SNR**.

**9. What Does This Code Do?**

1. Generates a 4x4 antenna array.
2. Simulates how signals arrive from different directions.
3. Calculates how the antennas combine these signals.
4. Adds noise to see how real-world conditions affect the received signals.
5. Stores the processed data for further analysis.

**10. Key Applications**

* This simulation is useful in:
  + Designing better wireless communication systems.
  + Creating radar systems that detect objects.
  + Improving satellite signal processing.

Let me know if you'd like any part clarified!

Here’s a high-level explanation of the provided code, structured so that a high school student can follow. It explains the background concepts, how the code works, and what the outputs mean.

**1. What Does This Code Do?**

This MATLAB program simulates a **rectangular antenna array** to study how signals behave when received from different directions. The antenna array is analyzed to determine how it responds to incoming signals, including:

* Adding noise to simulate real-world conditions.
* Calculating how signals from different directions (angles theta and phi) combine.
* Applying the **Discrete Fourier Transform (DFT)** to study the frequency characteristics of the received signal.

**2. Concepts in the Code**

**2.1. Antenna Array and Metasurface**

* **Antenna Array**: A grid of small antennas placed in a pattern (here, a 4x4 grid).
* **Metasurface**: A special type of antenna array designed to control how electromagnetic waves behave.
* Each antenna in the array contributes to the overall received signal, depending on its position and the direction from which the signal comes.

**2.2. Signal and Noise**

* **Signal**: A sinusoidal wave (simple wave that oscillates up and down smoothly) generated at 100 Hz.

Signal(t)=A⋅cos⁡(2πft)\text{Signal}(t) = A \cdot \cos(2 \pi f t)Signal(t)=A⋅cos(2πft)

* + AAA: Amplitude (how strong the signal is).
  + fff: Frequency (100 Hz).
  + ttt: Time vector.
* **Noise**: Unwanted disturbances added to the signal to simulate real-world imperfections.

**2.3. Array Factor**

The **Array Factor** measures how all the antennas combine their signals for a particular direction (angles theta and phi). It depends on:

1. **Positions of the antennas** in the grid.
2. **Phase shifts** of the signals arriving at different antennas.

**3. Code Walkthrough**

**3.1. Initializing the Antenna Array**

matlab

Copy code

Nx = 4; Ny = 4; % Grid dimensions

d = 0.5; % Spacing between antennas (in wavelengths)

lambda = 1; % Wavelength of the signal

k = 2 \* pi / lambda; % Wave number

* A 4x4 grid of antennas is created.
* Antennas are spaced 0.5 λ0.5 \, \lambda0.5λ apart, where λ\lambdaλ is the signal wavelength.
* k=2π/λk = 2\pi / \lambdak=2π/λ: Converts distances into phase shifts.

**3.2. Generating the Signal**

matlab

Copy code

fs = 1e3; % Sampling frequency (samples per second)

t = 0:1/fs:1; % Time vector (1 second)

A = 1; % Signal amplitude

signal = A \* cos(2 \* pi \* 100 \* t); % 100 Hz sinusoidal wave

* A sinusoidal signal at 100 Hz is created, sampled at 1000 Hz.

**3.3. Antenna Positions**

matlab

Copy code

[x, y] = meshgrid(0:Nx-1, 0:Ny-1); % Grid of positions

x = x \* d; y = y \* d; % Scale positions by antenna spacing

* The antennas are arranged in a grid, with positions stored in xxx and yyy.

**3.4. Looping Over Signal Conditions**

matlab

Copy code

for SNR = -5:2:5

for theta = -90:5:90

for phi = -90:5:90

received\_signal = calculate\_signal(SNR, theta, phi, Nx, Ny, x, y, signal, k, t);

received\_signal\_data = [received\_signal\_data; SNR, theta, phi, received\_signal];

end

end

end

* The program loops over:
  + **SNR** (signal-to-noise ratio): Simulates different levels of noise.
  + **Theta and Phi**: Angles describing the direction of the incoming signal.

**3.5. Calculating the Array Factor**

The **Array Factor** combines the contributions of all antennas:

matlab

Copy code

function Array\_factor = ArrayFactor(theta, phi, Nx, Ny, x, y, k)

Array\_factor = 0;

for i = 1:Nx

for j = 1:Ny

phase\_shift = k \* (x(i,j) \* sin(deg2rad(theta)) \* cos(deg2rad(phi)) + ...

y(i,j) \* sin(deg2rad(theta)) \* sin(deg2rad(phi)));

Array\_factor = Array\_factor + exp(1j \* phase\_shift);

end

end

end

* For each antenna, a **phase shift** is calculated based on its position and the incoming angles (theta, phi).
* These phase shifts are combined to calculate the overall response of the array.

**3.6. Adding Noise**

matlab

Copy code

noise = (1 / 10^(SNR / 10)) \* randn(size(received\_signal));

signal\_out = received\_signal + noise;

* Random noise is added to the signal based on the SNR. Higher SNR means less noise.

**3.7. Analyzing the Signal with DFT**

The **Discrete Fourier Transform (DFT)** is used to analyze the frequency content of the received signal:

matlab

Copy code

dft\_values = fft(first\_row\_signals); % Compute DFT

f = (0:length(dft\_values)-1) \* fs / length(dft\_values); % Frequency vector

plot(f, abs(dft\_values)); % Plot magnitude spectrum

* The DFT converts the signal from the time domain to the frequency domain.
* The plot shows the frequencies present in the signal and their magnitudes.

**4. Outputs**

1. **Received Signal Data**:
   * Combines the effects of direction (theta, phi), noise (SNR), and array factor.
2. **Magnitude Spectrum**:
   * Shows how much signal is present at different frequencies (like a fingerprint of the signal).

**5. Applications**

* **Wireless Communication**: Designing better antenna arrays for 5G networks.
* **Radar Systems**: Detecting objects by analyzing signals from multiple directions.
* **Satellite Communication**: Enhancing signal reception.

This code processes signal data from multiple rows (likely corresponding to different antenna elements or configurations) by performing an **8-point Radix-2 FFT (Fast Fourier Transform)** on the first eight signal samples of each row. The results for all rows are plotted on a single graph, with a legend to distinguish the FFT magnitudes of each row. Below is a detailed explanation, step by step.

**1. What is the Purpose of the Code?**

* **Objective**: Analyze the frequency content of the signals received by each row of the dataset (received\_signal\_data) and visualize how the FFT magnitude varies across rows.
* **Output**: A single graph showing FFT magnitudes for all rows, with individual rows labeled in the legend.

**2. Concepts in the Code**

**2.1. Fast Fourier Transform (FFT)**

* The **FFT** converts a signal from the **time domain** (variation over time) to the **frequency domain** (variation over frequencies).
* **8-point FFT**: Processes 8 samples of the signal, resulting in frequency components at 8 equally spaced points.

**2.2. Radix-2 FFT**

* A specific FFT algorithm optimized for data sizes that are powers of 2 (e.g., 8, 16, 32).
* **Advantages**: Fast computation, especially for real-time or large datasets.

**2.3. Frequency Vector**

* The frequency components for an 8-point FFT are: f=[0,fs8,2fs8,…,7fs8]f = \left[0, \frac{fs}{8}, \frac{2fs}{8}, \dots, \frac{7fs}{8}\right]f=[0,8fs​,82fs​,…,87fs​] where fsfsfs is the sampling frequency. These values correspond to the frequencies of the FFT output.

**2.4. Visualization**

* Each row of the dataset contributes one line to the plot, representing the FFT magnitude for that row.
* A legend labels each row, with a maximum of 50 rows shown for readability.

**3. Code Walkthrough**

**3.1. Initialize Variables**

matlab

Copy code

fft\_results = []; % Store FFT magnitudes for optional further processing

f = (0:7) \* (fs / 8); % Frequency vector for 8 points

* fft\_results: Stores the magnitudes of the FFT results for all rows.
* f: Frequency vector corresponding to the FFT output.

**3.2. Plot Preparation**

matlab

Copy code

figure; % Create a new figure

hold on; % Allow multiple lines on the same plot

graphics\_objects = gobjects(size(received\_signal\_data, 1), 1); % Placeholder for legend handles

* graphics\_objects: Array to store graphics objects (one for each row) for adding a legend later.

**3.3. Loop Through All Rows**

matlab

Copy code

for row = 1:size(received\_signal\_data, 1)

first\_8\_points = received\_signal\_data(row, 4:11); % Extract first 8 signal points

fft\_values = fft(first\_8\_points, 8); % Perform 8-point FFT

fft\_results = [fft\_results; abs(fft\_values)]; % Optional: Store magnitude of FFT

graphics\_objects(row) = plot(f, abs(fft\_values), 'DisplayName', ['Row ', num2str(row)]);

end

* For each row in received\_signal\_data:
  1. Extract the first 8 signal points (assumed to start at column 4).
  2. Compute the 8-point FFT using fft.
  3. Store the FFT magnitudes (optional).
  4. Plot the FFT magnitudes as a line on the graph.
  5. Label the line with the row number (Row X).

**3.4. Add Plot Details**

matlab

Copy code

title('8-Point Radix-2 FFT Magnitude (All Rows)');

xlabel('Frequency (Hz)');

ylabel('|FFT|');

grid on;

* Add a title, axis labels, and a grid for better visualization.

**3.5. Manage the Legend**

matlab

Copy code

if size(graphics\_objects, 1) > 50

legend(graphics\_objects(1:50), 'Location', 'best'); % Show first 50 rows only

disp('Only the first 50 rows are displayed in the legend due to space constraints.');

else

legend(graphics\_objects, 'Location', 'best'); % Show all rows

end

* If there are more than 50 rows, the legend only displays the first 50 for readability.
* Otherwise, all rows are included in the legend.

**4. Outputs**

1. **Plot**:
   * X-axis: Frequency components (fff, in Hz).
   * Y-axis: Magnitude of FFT (∣FFT∣|FFT|∣FFT∣).
   * Multiple lines, one for each row of received\_signal\_data, showing how the frequency content varies.
2. **Legend**:
   * Labels for up to 50 rows, identifying each line in the plot.
3. **FFT Results** (optional storage):
   * fft\_results: A matrix where each row contains the FFT magnitudes for a corresponding row in received\_signal\_data.

**5. Applications**

* **Signal Processing**: Analyze frequency responses of signals from different antennas or receivers.
* **Pattern Analysis**: Identify patterns in the frequency domain for different rows.
* **Diagnostics**: Compare signal quality or detect anomalies by inspecting FFT magnitudes.

This code provides a powerful way to visualize and analyze frequency-domain data across multiple signals efficiently. Let me know if you'd like further clarification!

This MATLAB code simulates a **rectangular antenna array metasurface** with noisy signals, processes the received signals into a format suitable for training a neural network, and then trains a feedforward artificial neural network (ANN) to predict system parameters like **SNR, θ, and φ**. Below is an **extremely detailed explanation** of the code and its components.

**1. Objectives**

The code performs three primary tasks:

1. **Signal Simulation**:
   * Simulates signals received by a rectangular metasurface antenna array for varying system parameters such as Signal-to-Noise Ratio (SNR), angle of arrival θ\thetaθ, and azimuth angle ϕ\phiϕ.
   * Adds noise to simulate real-world conditions.
2. **Data Preparation**:
   * Prepares the signal data (real and imaginary components) for use in machine learning.
3. **ANN Training**:
   * Trains a feedforward neural network to predict SNR\text{SNR}SNR, θ\thetaθ, and ϕ\phiϕ from the processed signals.

**2. Signal Simulation**

**2.1. Rectangular Antenna Array**

The metasurface is modeled as a rectangular grid of elements:

* Nx=4Nx = 4Nx=4: Number of elements in the x-direction.
* Ny=4Ny = 4Ny=4: Number of elements in the y-direction.
* d=0.5d = 0.5d=0.5: Distance between elements in terms of the wavelength λ\lambdaλ.
* **Grid Creation**:

matlab

Copy code

[x, y] = meshgrid(0:Nx-1, 0:Ny-1);

x = x \* d;

y = y \* d;

* + Creates a 4×4 grid of element positions based on NxNxNx, NyNyNy, and ddd.

**2.2. Incoming Signal**

A simple sinusoidal wave is generated as the incoming signal:

matlab

Copy code

signal = A \* cos(2 \* pi \* 100 \* t);

* **Amplitude**: A=1A = 1A=1.
* **Frequency**: 100 Hz100 \, \text{Hz}100Hz.
* **Time Vector**: ttt is sampled at fs=1000 Hzfs = 1000 \, \text{Hz}fs=1000Hz for one second.

**2.3. Loop Over Parameters**

The code iterates through different values of SNR, θ\thetaθ, and ϕ\phiϕ to simulate the received signals under various conditions:

* SNR\text{SNR}SNR: -5 dB to +5 dB (low-noise to high-noise environments).
* θ\thetaθ: -90° to +90° (elevation angles).
* ϕ\phiϕ: -90° to +90° (azimuth angles).

For each combination:

1. The **array factor** is calculated (representing the combined response of the antenna array).
2. Noise is added based on the SNR.
3. The received signal is stored.

**2.3.1. Array Factor Calculation**

The **array factor** describes the contribution of all antenna elements to the received signal. For each element (i,j)(i, j)(i,j):

matlab

Copy code

phase\_shift = k \* (x(i,j) \* sin(deg2rad(theta)) \* cos(deg2rad(phi)) + ...

y(i,j) \* sin(deg2rad(theta)) \* sin(deg2rad(phi)));

Array\_factor = Array\_factor + exp(1j \* phase\_shift);

* **Phase Shift**: phase\_shift=k(xijsin⁡θcos⁡ϕ+yijsin⁡θsin⁡ϕ)\text{phase\\_shift} = k \left( x\_{ij} \sin\theta \cos\phi + y\_{ij} \sin\theta \sin\phi \right)phase\_shift=k(xij​sinθcosϕ+yij​sinθsinϕ)
  + k=2π/λk = 2\pi / \lambdak=2π/λ is the wave number.
  + xij,yijx\_{ij}, y\_{ij}xij​,yij​: Element positions.

**2.3.2. Noise Addition**

The received signal is adjusted for SNR using the equation:

matlab

Copy code

noise = (1 / 10^(SNR / 10)) \* randn(size(received\_signal));

signal\_out = received\_signal + noise;

* Noise power is inversely proportional to 10SNR/1010^{\text{SNR}/10}10SNR/10, simulating a real-world environment.

**2.4. Data Storage**

The generated dataset is structured as:

* Columns 1–3: SNR,θ,ϕ\text{SNR}, \theta, \phiSNR,θ,ϕ.
* Remaining columns: Simulated noisy signal values.

**3. Data Preparation for ANN**

The dataset is split into **inputs** and **outputs**:

* **Outputs**:

matlab

Copy code

outputs = received\_signal\_data(:, 1:3)';

* + Transposed for ANN format: Each column corresponds to a data sample.
  + SNR,θ,ϕ\text{SNR}, \theta, \phiSNR,θ,ϕ are the targets.
* **Inputs**:

matlab

Copy code

inputs = received\_signal\_data(:, 4:end)';

* + Real and imaginary parts are separated and combined:

matlab

Copy code

inputs\_real = real(inputs);

inputs\_imag = imag(inputs);

inputs\_processed = [inputs\_real; inputs\_imag];

**4. ANN Training**

**4.1. Feedforward Network**

The neural network is defined as:

matlab

Copy code

hiddenLayerSize = 200; % Reduced number of neurons

net = feedforwardnet(hiddenLayerSize);

* **Architecture**:
  + Input layer: Combined real and imaginary signal data.
  + Hidden layer: 200 neurons.
  + Output layer: 3 neurons (SNR, θ\thetaθ, ϕ\phiϕ).

**4.2. Training in Batches**

The training process is performed in smaller batches to handle large datasets:

matlab

Copy code

batchSize = 1000;

for batch = 1:numBatches

batchInputs = inputs\_processed(:, batchStart:batchEnd);

batchOutputs = outputs(:, batchStart:batchEnd);

[net, tr] = train(net, batchInputs, batchOutputs);

end

* **Batch Size**: 1000 samples per batch.
* **Training Algorithm**: Resilient Backpropagation (trainrp), suitable for moderate-sized datasets.

**5. ANN Testing and Evaluation**

**5.1. Prediction**

The trained ANN predicts outputs for the input dataset:

matlab

Copy code

predicted\_outputs = net(inputs\_processed);

**5.2. Performance Evaluation**

The performance is evaluated using the **mean squared error (MSE)**:

matlab

Copy code

performance = perform(net, outputs, predicted\_outputs);

Visualization tools:

* plotperform(tr): Plots training performance over epochs.
* plotregression(outputs, predicted\_outputs): Displays a regression plot comparing true and predicted values.

**6. Saving the Model**

The trained ANN is saved for future use:

matlab

Copy code

save('trained\_ANN.mat', 'net');

**7. Outputs**

1. **Trained Neural Network**:
   * Capable of predicting SNR, θ\thetaθ, and ϕ\phiϕ for new input signals.
2. **Performance Metrics**:
   * MSE and regression plots.
3. **Saved Model**:
   * File trained\_ANN.mat for reuse.

**Applications**

1. **Wireless Communication**: Optimize antenna arrays for beamforming.
2. **Machine Learning**: Train models for signal processing tasks.
3. **Signal Diagnostics**: Predict system parameters in noisy environments.

Let me know if you'd like further clarifications or modifications!