Project Title: GUI-Based Adaptive Beamforming System Using LMS Algorithm in MATLAB

1. **Introduction:** Adaptive beamforming is a signal processing technique used in sensor arrays for directional signal transmission or reception. It enhances the signal quality from a desired direction while suppressing signals and interference from other directions. The Least Mean Squares (LMS) algorithm is a widely used adaptive filtering algorithm due to its simplicity and effectiveness. This project aims to simulate and visualize the behavior of an adaptive beamforming system using LMS via a MATLAB App Designer GUI.

2. **Objective:** To design and implement a user-friendly MATLAB GUI application for simulating adaptive beamforming using the LMS algorithm with configurable input parameters and real-time visualization of error magnitude and beam pattern.

3. **Key Features:**

   - Fully interactive GUI using MATLAB App Designer.
   - User-defined parameters: number of antennas, desired signal angle, interferer angles, SNRs, and LMS step size.
   - Real-time plotting of convergence (error magnitude) and directional response (beam pattern).
   - Visual confirmation of null placement at interferer directions and beam alignment toward the desired direction.

4. **Theory Background:**

   - **Antenna Array:** A linear array of antennas receives a combination of desired signal and interferers.
   - **Steering Vector:** Represents phase shifts across elements to focus on specific angles.
   - **Signal Model:** $x(n) = a_d s_d(n) + \sum_k a_i(k) s_i(k,n) + noise$
   - **LMS Algorithm:** $w(n+1) = w(n) + \mu x(n) e^*(n)$ $e(n) = d(n) - w^H(n) x(n)$ Where:
   - $w(n)$: weight vector
   - $x(n)$: input signal vector
   - $d(n)$: desired signal
   - $e(n)$: error signal
   - μ : step size

5. **System Design:**

   - **Input Components:** Numeric fields for antenna count, angles, SNRs, and step size.
   - **Processing:**
   - Signal generation for desired and interferers.
   - Steering vector computation.
   - Adaptive weight update using LMS.
   - **Outputs:**
   - Plot of $|e[n]|$ over time to show convergence.
   - Beam pattern plot to show angular response.

6. **GUI Implementation Highlights:**

   - Developed with App Designer in MATLAB.

   - Callback functions to process user input and update plots.
   - Axes components for dynamic real-time plotting.

- Handles invalid input gracefully (e.g., text instead of numeric).

7. **Sample Parameters Used in Demo:**

   - Number of Antennas: 8
   - Desired Angle: 30°
   - Interferer Angles: 60°, 90°
   - Desired SNR: 20 dB
   - Interferer SNR: 10 dB
   - Step Size: 0.005

8. **Results & Analysis:**

   - **Error Plot:** Shows exponential decay and stabilization, indicating successful LMS adaptation.
   - **Beam Pattern:** Peak beam at 30°, nulls at 60° and 90°, confirming effective interference suppression.
   - **System Responsiveness:** The GUI responds in real time, making parameter sweeps and analysis intuitive.

9. **Applications:**

   - Radar and sonar systems
   - Wireless communications (5G/6G)
   - Smart antenna systems
   - Satellite communications

10. **Conclusion:** This project demonstrates an advanced yet intuitive tool for exploring adaptive beamforming. The LMS-based approach effectively adapts to interference, and the GUI framework offers an ideal educational and research simulation environment. The project is highly suitable for inclusion in an ECE student portfolio or resume.

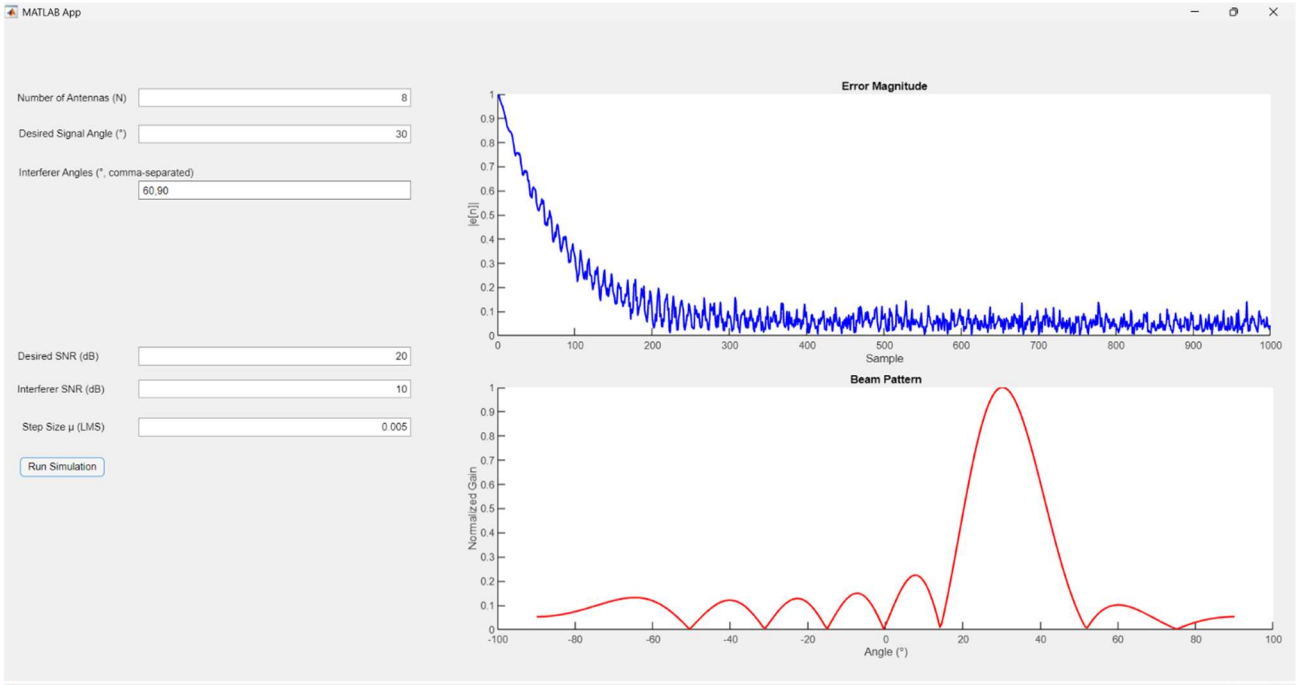
11. **Future Scope:**

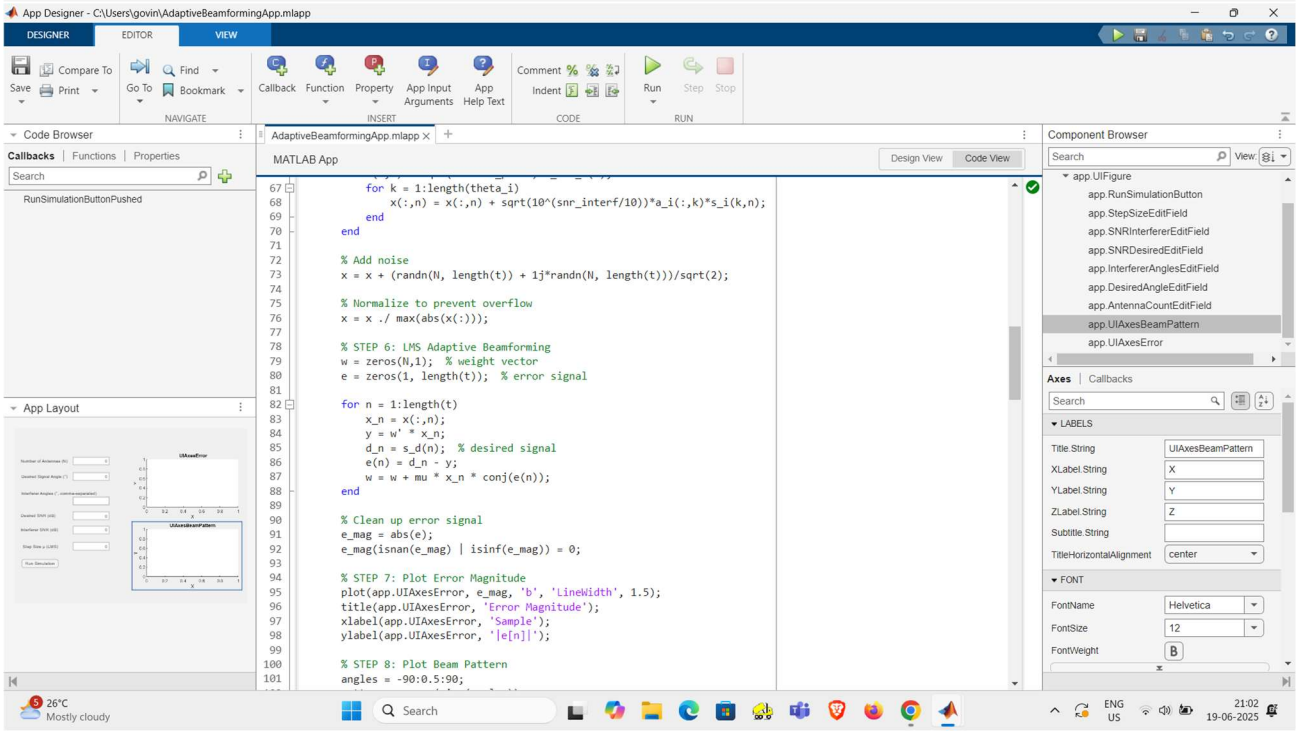
    - Add support for non-linear arrays.
    - Extend to RLS and other adaptive algorithms.
    - Real-time hardware interfacing using SDR platforms (e.g., USRP).
    - Export data and figures as PDF/CSV.

12. **Tools Used:**

    - MATLAB R2023a+
    - MATLAB App Designer
    - Signal Processing Toolbox

13. Screenshots





RunSimulationButtonPushed(app, event) function code :

```
% STEP 1: Read values from UI
   N = app.AntennaCountEditField.Value;
   theta_d = app.DesiredAngleEditField.Value;
   theta_i = str2num(app.InterfererAnglesEditField.Value); %#ok<ST2NM>
   snr_dB = app.SNRDesiredEditField.Value;
   snr_interf = app.SNRInterfererEditField.Value;
   mu = app.StepSizeEditField.Value;

   % STEP 2: Simulation setup
   d = 0.5;            % inter-element spacing (λ/2)
   fs = 1000;          % sampling frequency
   T = 1;              % duration
   t = 0:1/fs:T-1/fs;    % time vector

   % STEP 3: Steering vector function
   steering_vector = @(theta) exp(-1j*2*pi*d*(0:N-1)'*sin(deg2rad(theta)));

   % STEP 4: Desired and Interferer signals
   a_d = steering_vector(theta_d);
   a_i = zeros(N, length(theta_i));
   for k = 1:length(theta_i)
      a_i(:,k) = steering_vector(theta_i(k));
   end

   f_d = 100;
   s_d = exp(1j*2*pi*f_d*t); % desired signal

   f_i = linspace(200, 300, length(theta_i));
   s_i = zeros(length(theta_i), length(t));
   for k = 1:length(theta_i)
      s_i(k,:) = exp(1j*2*pi*f_i(k)*t);
   end

   % STEP 5: Input signal at antenna array
   x = zeros(N, length(t));
   desired_power = 10^(snr_dB/10);
   for n = 1:length(t)
      x(:,n) = sqrt(desired_power)*a_d*s_d(n);
      for k = 1:length(theta_i)
         x(:,n) = x(:,n) + sqrt(10^(snr_interf/10))*a_i(:,k)*s_i(k,n);
      end
   end

   % Add noise
   x = x + (randn(N, length(t)) + 1j*randn(N, length(t)))/sqrt(2);

   % Normalize to prevent overflow
   x = x ./ max(abs(x(:)));

   % STEP 6: LMS Adaptive Beamforming
   w = zeros(N,1);  % weight vector
   e = zeros(1, length(t));  % error signal
```

```
for n = 1:length(t)
    x_n = x(:,n);
    y = w' * x_n;
    d_n = s_d(n);  % desired signal
    e(n) = d_n - y;
    w = w + mu * x_n * conj(e(n));
end

% Clean up error signal
e_mag = abs(e);
e_mag(isnan(e_mag) | isinf(e_mag)) = 0;

% STEP 7: Plot Error Magnitude
plot(app.UIAxesError, e_mag, 'b', 'LineWidth', 1.5);
title(app.UIAxesError, 'Error Magnitude');
xlabel(app.UIAxesError, 'Sample');
ylabel(app.UIAxesError, '|e[n]|');

% STEP 8: Plot Beam Pattern
angles = -90:0.5:90;
pattern = zeros(size(angles));
for idx = 1:length(angles)
    a_theta = steering_vector(angles(idx));
    pattern(idx) = abs(w' * a_theta);
end
pattern = pattern / max(pattern);  % Normalize

plot(app.UIAxesBeamPattern, angles, pattern, 'r', 'LineWidth', 1.5);
title(app.UIAxesBeamPattern, 'Beam Pattern');
xlabel(app.UIAxesBeamPattern, 'Angle (°)');
ylabel(app.UIAxesBeamPattern, 'Normalized Gain');
```