```
clear; close all; clc;

% Simulation Parameters
dt = 0.1;          % Sampling time
T = 10;           % Simulation duration [s]
time = 0:dt:T;     % Time vector
n = 6;            % State dimension
p = 5;            % Measurement dimension
M = 400;          % Number of Monte Carlo runs

% Process and Measurement Noise Covariances
Q = diag([0.01, 0.01, 0.01, 0.01, 0.01, 0.01]); % Process noise covariance
R = diag([0.1, 0.1, 0.01, 0.1, 0.01]);                    % Measurement noise
covariance

% Initial Perturbation and Covariance
perturbation = [0.1; 0.1; 0.05; 0.1; 0.1; 0.05];
P0 = diag([0.1, 0.1, 0.1, 0.1, 0.1, 0.1]);

% Nominal Trajectory
xi_g_nom = @(t) (1/(2*tan(pi/18))) * (20*tan(pi/18) + 1 - cos(4*tan(-pi/18)*t));
eta_g_nom = @(t) (1/(2*tan(pi/18))) * sin(4*tan(-pi/18)*t);
theta_g_nom = @(t) pi/2 + 4*tan(-pi/18)*t;
xi_a_nom = @(t) (1/pi)*(300 - 60*t - 300*cos(pi/25*t));
eta_a_nom = @(t) (300/pi)*sin(pi/25*t);
theta_a_nom = @(t) wrapToPi(-pi/2 + pi/25*t);
nominal_state = @(t) [xi_g_nom(t); eta_g_nom(t); theta_g_nom(t); xi_a_nom(t);
eta_a_nom(t); theta_a_nom(t)];

% Nonlinear Dynamics
f = @(x) [2*cos(x(3)); 2*sin(x(3)); (2/0.5)*tan(-pi/18); 12*cos(x(6));
12*sin(x(6)); pi/25];

% Linearized System Dynamics (Continuous-Time Jacobians)
A = @(x) [0, 0, -2*sin(x(3)), 0, 0, 0;
          0, 0,  2*cos(x(3)), 0, 0, 0;
          0, 0,  0,           0, 0, 0;
          0, 0,  0,           0, 0, -12*sin(x(6));
          0, 0,  0,           0, 0,  12*cos(x(6));
          0, 0,  0,           0, 0,  0];

C = @(x) [1, 0, 0, -1, 0, 0;
          0, 1, 0, 0, -1, 0;
          0, 0, 1, 0, 0, -1;
          ((x(5) - x(2)) / sqrt((x(5) - x(2))^2 + (x(4) - x(1))^2)), ...
          (-(x(4) - x(1)) / sqrt((x(5) - x(2))^2 + (x(4) - x(1))^2)), 0, ...
          (-(x(5) - x(2)) / sqrt((x(5) - x(2))^2 + (x(4) - x(1))^2)), ...
          ((x(4) - x(1)) / sqrt((x(5) - x(2))^2 + (x(4) - x(1))^2)), 0;
          ((x(4) - x(1)) / ((x(5) - x(2))^2 + (x(4) - x(1))^2)), ...
          ((x(5) - x(2)) / ((x(5) - x(2))^2 + (x(4) - x(1))^2)), 0, ...
          ((x(4) - x(1)) / ((x(5) - x(2))^2 + (x(4) - x(1))^2)), ...
          ((x(5) - x(2)) / ((x(5) - x(2))^2 + (x(4) - x(1))^2)), 0];
```

```
% Discretize System
F_k = @(x) eye(n) + dt * A(x);
G_k = @(x) dt * [cos(x(3)), 0, 0, 0;
                 sin(x(3)), 0, 0, 0;
                 (2*tan(-pi/18))/0.5, 12/(0.5*cos(-pi/18)^2), 0, 0;
                 0, 0, cos(x(6)), 0;
                 0, 0, sin(x(6)), 0;
                 0, 0, 0, 1];

% Storage for NEES and NIS
NEESsshist = zeros(length(time), M);
NISsshist = zeros(length(time), M);

% Monte Carlo Simulation
figure(1);
h1 = subplot(2, 1, 1); % NEES plot
h2 = subplot(2, 1, 2); % NIS plot
hold(h1, 'on'); hold(h2, 'on');

alpha = 0.05;
nees_bounds = [chi2inv(alpha/2, n), chi2inv(1-alpha/2, n)];
nis_bounds = [chi2inv(alpha/2, p), chi2inv(1-alpha/2, p)];
disp('NEES Bounds:'), disp(nees_bounds);
disp('NIS Bounds:'), disp(nis_bounds);
for m = 1:M
    % Simulate Ground Truth Trajectory
    x_true = nominal_state(0) + perturbation;
    x_true_trajectory = zeros(n, length(time));
    y_meas_trajectory = zeros(p, length(time));

    for k = 1:length(time)
        % Store Ground Truth
        x_true_trajectory(:, k) = x_true;

        % Generate Noisy Measurement
        C_k = C(x_true);
        noise = mvnrnd(zeros(p, 1), R)'; % Ensure noise is a column vector
        y_meas_trajectory(:, k) = C_k * x_true + noise;

        % Propagate True State (Euler Integration)
        x_true = x_true + dt * f(x_true) + mvnrnd(zeros(n, 1), Q)';
    end

    % Initialize LKF
    x_hat = nominal_state(0) + perturbation; % Initial state estimate
    P = P0;                                  % Initial covariance

    % Apply LKF
    for k = 1:length(time)
        % Prediction Step
        F = F_k(nominal_state(time(k)));
        G = G_k(nominal_state(time(k)));
        u = [2; -pi/18; 12; pi/25]; % Control input vector
        x_hat = F * x_hat + G * u;
```

```
        P = F * P * F' + Q;

        % Update Step
        C_k = C(x_hat);
        S = C_k * P * C_k' + R; % Innovation covariance
        K = P * C_k' / S;       % Kalman gain
        y_innov = y_meas_trajectory(:, k) - C_k * x_hat; % Innovation
        x_hat = x_hat + K * y_innov;
        P = (eye(n) - K * C_k) * P;

        % NEES and NIS
        NEESsshist(k, m) = (x_true_trajectory(:, k) - x_hat)' * inv(P) *
(x_true_trajectory(:, k) - x_hat);
        NISsshist(k, m) = y_innov' * inv(S) * y_innov;
    end
% Update NEES and NIS Visualization
    subplot(h1);
    plot(mean(NEESsshist(:, 1:m), 2), 'r', 'LineWidth', 1.5); hold on;
    xlabel('Time Step'); ylabel('NEES'); title('Dynamic NEES Visualization');
grid on;
    plot(k, nees_bounds(1), 'r--', 'LineWidth', 1.2); % Lower bound
    plot(k, nees_bounds(2), 'r--', 'LineWidth', 1.2); % Upper bound
    xlabel('Time Step'); ylabel('NEES'); title('Dynamic NEES Visualization');
grid on;

    subplot(h2);
    plot(mean(NISsshist(:, 1:m), 2), 'b', 'LineWidth', 1.5); hold on;
    xlabel('Time Step'); ylabel('NIS'); title('Dynamic NIS Visualization'); grid
on;
    plot(k, nis_bounds(1), 'b--', 'LineWidth', 1.2); % Lower bound
    plot(k, nis_bounds(2), 'b--', 'LineWidth', 1.2); % Upper bound
    xlabel('Time Step'); ylabel('NIS'); title('Dynamic NIS Visualization'); grid
on;


    drawnow;
end

% Final NEES and NIS Bounds and Averages
nees_avg = mean(NEESsshist, 2);
nis_avg = mean(NISsshist, 2);

% Final Plot for NEES
figure;
subplot(2, 1, 1);
plot(1:length(time), nees_avg, 'ro', 'MarkerSize', 6, 'LineWidth', 1.5); % Red
points
hold on;
plot(1:length(time), nees_bounds(1) * ones(size(time)), '--r', 'LineWidth',
1.5); % Lower bound
plot(1:length(time), nees_bounds(2) * ones(size(time)), '--r', 'LineWidth',
1.5); % Upper bound
title('NEES Estimation Results');
xlabel('Time Step, k');
```

```
ylabel('NEES Statistic, \bar{\epsilon}_x');
grid on;

% Final Plot for NIS
subplot(2, 1, 2);
plot(1:length(time), nis_avg, 'bo', 'MarkerSize', 6, 'LineWidth', 1.5); % Blue
points
hold on;
plot(1:length(time), nis_bounds(1) * ones(size(time)), '--b', 'LineWidth', 1.5);
% Lower bound
plot(1:length(time), nis_bounds(2) * ones(size(time)), '--b', 'LineWidth', 1.5);
% Upper bound
title('NIS Estimation Results');
xlabel('Time Step, k');
ylabel('NIS Statistic, \bar{\epsilon}_y');
grid on;
```