```
% Extended Kalman Filter (EKF) with NEES, NIS, and Monte Carlo Simulations

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 = 10*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, u) [u(1)*cos(x(3)); u(1)*sin(x(3)); (u(1)/0.5)*tan(u(2));
u(3)*cos(x(6)); u(3)*sin(x(6)); u(4)];

% Nonlinear Measurement Model
h = @(x) [x(1) - x(4); x(2) - x(5); x(3) - x(6); ...
          sqrt((x(4) - x(1))^2 + (x(5) - x(2))^2); ...
          atan2(x(5) - x(2), x(4) - x(1))];

% Jacobian of Dynamics
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];

B = @(x) [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];
```

```
% Discretize Jacobians
F_k = @(x) eye(n) + dt * A(x);
G_k = @(x) dt * B(x);

% Jacobian of Measurement Model
H = @(x) [1, 0, 0, -1, 0, 0;
          0, 1, 0, 0, -1, 0;
          0, 0, 1, 0, 0, -1;
          (x(1)-x(4))/sqrt((x(1)-x(4))^2 + (x(2)-x(5))^2), ...
          (x(2)-x(5))/sqrt((x(1)-x(4))^2 + (x(2)-x(5))^2), 0, ...
          -(x(1)-x(4))/sqrt((x(1)-x(4))^2 + (x(2)-x(5))^2), ...
          -(x(2)-x(5))/sqrt((x(1)-x(4))^2 + (x(2)-x(5))^2), 0;
          (x(2)-x(5))/((x(1)-x(4))^2 + (x(2)-x(5))^2), ...
          (x(1)-x(4))/((x(1)-x(4))^2 + (x(2)-x(5))^2), 0, ...
          -(x(2)-x(5))/((x(1)-x(4))^2 + (x(2)-x(5))^2), ...
          -(x(1)-x(4))/((x(1)-x(4))^2 + (x(2)-x(5))^2), 0];

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

% Control Input
u = [2; -pi/18; 12; pi/25];

% Monte Carlo Simulations
for m = 1:M
    % Simulate Ground Truth Trajectory
    x_true = [1; 1; 0.1; 2; 2; 0.2] + perturbation; % True state
    x_hat = zeros(n, 1);                            % Initial state estimate
    P = P0;                                        % Initial covariance

    for k = 1:length(time)
        % True state propagation
        F = F_k(x_true);
        G = G_k(x_true);
        x_true = F * x_true + G * u + mvnrnd(zeros(n, 1), Q)';

        % Generate measurement
        y = h(x_true) + mvnrnd(zeros(p, 1), R)';

        % Prediction step
        F = F_k(x_hat);               % Linearized dynamics
        G = G_k(x_hat);               % Linearized control input
        x_hat = F * x_hat + G * u;    % Predicted state
        P = F * P * F' + Q;           % Predicted covariance

        % Update step
        H_k = H(x_hat);               % Linearized measurement model
        y_hat = h(x_hat);             % Predicted measurement
        S = H_k * P * H_k' + R;       % Innovation covariance
        K = P * H_k' / S;             % Kalman gain
        y_innov = y - y_hat;          % Innovation
        x_hat = x_hat + K * y_innov;  % State update
```

```
        P = (eye(n) - K * H_k) * P;   % Covariance update

        % NEES and NIS
        NEESsshist(k, m) = (x_true - x_hat)' * inv(P) * (x_true - x_hat);
        NISsshist(k, m) = y_innov' * inv(S) * y_innov;
    end
end

% Calculate Averages
nees_avg = mean(NEESsshist, 2);
nis_avg = mean(NISsshist, 2);

% Chi-squared Bounds
alpha = 0.05;
nees_bounds = [chi2inv(alpha/2, n) / M, chi2inv(1 - alpha/2, n) / M];
nis_bounds = [chi2inv(alpha/2, p) / M, chi2inv(1 - alpha/2, p) / M];

% Plot Results
figure;

% NEES Plot
subplot(2, 1, 1);
plot(1:length(time), nees_avg, 'r-o', 'LineWidth', 1.5); hold on;
%for m = 1:M
  %  plot(1:length(time), NEESsshist(:, m), 'r-', 'LineWidth', 0.2);
%end
yline(nees_bounds(1), '--r', 'Lower Bound');
yline(nees_bounds(2), '--r', 'Upper Bound');
title('NEES Estimation Results');
xlabel('Time Step, k');
ylabel('NEES');
grid on;

% NIS Plot
subplot(2, 1, 2);
plot(1:length(time), nis_avg, 'b-o', 'LineWidth', 1.5); hold on;
%for m = 1:M
   % plot(1:length(time), NISsshist(:, m), 'b-', 'LineWidth', 0.2);
   %end
yline(nis_bounds(1), '--r', 'Lower Bound');
yline(nis_bounds(2), '--r', 'Upper Bound');
title('NIS Estimation Results');
xlabel('Time Step, k');
ylabel('NEES');
grid on;
```