Table of Contents

```
clc; clear; close;

% Load all data files from the input directory, the variables will be named
after the file names
load_directory_data('input');
load_directory_data('ref', '_ref');
if ~exist('output', 'dir')
    mkdir('output');
end

% Constants
C = constants();

% Format input data
x_GPS = permute(cat(3, rx_gps, ry_gps, rz_gps, vx_gps, vy_gps, vz_gps), [3,
1, 2]); % 6 x n_time x N_sats
x_ref = [rx_ref'; ry_ref'; rz_ref'; vx_ref'; vy_ref'; vz_ref']; % 6 x n_time

% Q5. Initial propagation
x1 = [rx_ref(1); ry_ref(1); rz_ref(1); vx_ref(1); vy_ref(1); vz_ref(1)];
[x2, Phi2] = propagation(x1, eye(6), t(2)-t(1)); % Propagate to t2

fprintf('Q5:\n');
fprintf('   Propagated state at t2:\n');
fprintf('   r(t2) = %.6f %.6f %.6f km \n', x2(1:3));
fprintf('   v(t2) = %.6f %.6f %.6f km/s \n', x2(4:6));
fprintf('   STM at t2:\n');
fprintf('   Phi_22(t2, t1) = %.10f \n', Phi2(2,2));
fprintf('   Phi_15(t2, t1) = %.10f \n', Phi2(1,5));
fprintf('   Phi_36(t2, t1) = %.10f \n', Phi2(3,6));
fprintf('   Phi_52(t2, t1) = %.10f \n', Phi2(5,2));

% EKF without process noise
sigma_rho = 3e-3;    % km
sigma_pos = 2e-3;    % km
sigma_vel = 0.1e-3; % km/s
rho_r_v = 0.7;       % correlation coefficient between position and velocity

% Construct P0 (6x6 covariance matrix)
P0 = diag([sigma_pos^2, sigma_pos^2, sigma_pos^2, ...
           sigma_vel^2, sigma_vel^2, sigma_vel^2]);
P0(1,4) = rho_r_v * sigma_pos * sigma_vel;
P0(4,1) = P0(1,4);
P0(2,5) = rho_r_v * sigma_pos * sigma_vel;
```

```
P0(5,2) = P0(2,5);
P0(3,6) = rho_r_v * sigma_pos * sigma_vel;
P0(6,3) = P0(3,6);

% Run EKF without process noise
[x_est_ekf_nn, P_est_ekf_nn, ~] = ekf(t, x1, P0, sigma_rho, CA_range, PRN_ID, x_GPS); % No process noise
d_r_ekf_nn = vecnorm(x_est_ekf_nn(1:3, :) - x_ref(1:3, :), 2, 1); % Position error over time no process noise
d_v_ekf_nn = vecnorm(x_est_ekf_nn(4:6, :) - x_ref(4:6, :), 2, 1); % Velocity error over time no process noise
sigma_r_ekf_nn = sqrt(squeeze(P_est_ekf_nn(1,1,:) + P_est_ekf_nn(2,2,:) + P_est_ekf_nn(3,3,:))); % Position 3-sigma no process noise
sigma_v_ekf_nn = sqrt(squeeze(P_est_ekf_nn(4,4,:) + P_est_ekf_nn(5,5,:) + P_est_ekf_nn(6,6,:))); % Velocity 3-sigma no process noise

fprintf('\nQ6:\n');
fprintf('   EKF without process noise:\n');

% Position table
fprintf('   Epoch 10: r = (%.6f, %.6f, %.6f) km\n', x_est_ekf_nn(1:3, 10));
fprintf('   Epoch 20: r = (%.6f, %.6f, %.6f) km\n', x_est_ekf_nn(1:3, 20));
fprintf('   Epoch 30: r = (%.6f, %.6f, %.6f) km\n', x_est_ekf_nn(1:3, 30));

% Velocity table
fprintf('   Epoch 10: v = (%.6f, %.6f, %.6f) km/s\n', x_est_ekf_nn(4:6, 10));
fprintf('   Epoch 20: v = (%.6f, %.6f, %.6f) km/s\n', x_est_ekf_nn(4:6, 20));
fprintf('   Epoch 30: v = (%.6f, %.6f, %.6f) km/s\n', x_est_ekf_nn(4:6, 30));

% Run EKF with process noise
sigma_a = 0.01e-3; % km
sigma_b = 0.01e-3; % km/s
Q = [(sigma_a^2)*eye(3), zeros(3); zeros(3), (sigma_b^2)*eye(3)]; % Process noise covariance matrix
[x_est_ekf_wn, P_est_ekf_wn, obs_res_wn] = ekf(t, x1, P0, sigma_rho, CA_range, PRN_ID, x_GPS, Q); % With process noise
d_r_ekf_wn = vecnorm(x_est_ekf_wn(1:3, :) - x_ref(1:3, :), 2, 1); % Position error over time with process noise
d_v_ekf_wn = vecnorm(x_est_ekf_wn(4:6, :) - x_ref(4:6, :), 2, 1); % Velocity error over time with process noise
sigma_r_ekf_wn = sqrt(squeeze(P_est_ekf_wn(1,1,:) + P_est_ekf_wn(2,2,:) + P_est_ekf_wn(3,3,:))); % Position 3-sigma with process noise
sigma_v_ekf_wn = sqrt(squeeze(P_est_ekf_wn(4,4,:) + P_est_ekf_wn(5,5,:) + P_est_ekf_wn(6,6,:))); % Velocity 3-sigma with process noise
```

Plotting

```
set(groot, 'defaultTextInterpreter', 'latex');
set(groot, 'defaultLegendInterpreter', 'latex');
set(groot, 'defaultAxesTickLabelInterpreter', 'latex');
set(groot, 'defaultAxesFontSize', 16);
set(groot, 'defaultLegendFontSize', 16);
set(groot, 'defaultAxesLabelFontSizeMultiplier', 20/16);
set(groot, 'defaultAxesTitleFontSizeMultiplier', 22/16);
set(groot, 'defaultAxesLineWidth', 1.2);
set(groot, 'defaultLineLineWidth', 1.5);
set(groot, 'defaultAxesGridAlpha', 0.3);
set(groot, 'defaultAxesMinorGridAlpha', 0.15);
set(groot, 'DefaultFigureVisible', 'off');

% Q7
fig = figure('Position',[200 200 1600 900]);
yyaxis left
h1 = plot(t-t(1), d_r_ekf_nn*1000);
ylabel('Position Error [m]');
yyaxis right
h2 = plot(t-t(1), d_v_ekf_nn*1000);
ylabel('Velocity Error [m/s]');
xlabel('Elapsed Time [s]');
title('EKF Position and Velocity Error without Process Noise');
grid on;
saveas(fig, fullfile('output','error_ekf_nn.png'));
close(fig);

% Q8
fig = figure('Position',[200 200 1600 900]);
yyaxis left
h1 = plot(t-t(1), sigma_r_ekf_nn*1000);
set(gca, 'YScale', 'log');
ylabel('Position $\sigma$ [m]');
yyaxis right
h2 = plot(t-t(1), sigma_v_ekf_nn*1000);
set(gca, 'YScale', 'log');
ylabel('Velocity $\sigma$ [m/s]');
xlabel('Elapsed Time [s]');
title('EKF Position and Velocity Standard Deviation without Process Noise');
grid on;
saveas(fig, fullfile('output','sigma_ekf_nn.png'));
close(fig);

% Q14
fig = figure('Position',[200 200 1600 1000]);
subplot(2,1,1);
yyaxis left
h1 = plot(t-t(1), d_r_ekf_wn*1000);
hold on;
h2 = plot(t-t(1), d_r_ekf_nn*1000, '--', Color = h1.Color);
ylabel('Position Error [m]');
yyaxis right
h3 = plot(t-t(1), d_v_ekf_wn*1000);
hold on;
h4 = plot(t-t(1), d_v_ekf_nn*1000, '--', Color = h3.Color);
ylabel('Velocity Error [m/s]');
xlabel('Elapsed Time [s]');
% Empty plots to show legend in black
e1 = plot(nan, nan, '-k', 'DisplayName', 'No Process Noise');
e2 = plot(nan, nan, '--k', 'DisplayName', 'With Process Noise');
```

```
legend([e1, e2], 'Location', 'best');
title('EKF Position and Velocity Error');
grid on;
subplot(2,1,2);
yyaxis left
h5 = plot(t-t(1), sigma_r_ekf_wn*1000);
hold on;
h6 = plot(t-t(1), sigma_r_ekf_nn*1000, '--', Color = h5.Color);
set(gca, 'YScale', 'log');
ylabel('Position $\sigma$ [m]');
yyaxis right
h7 = plot(t-t(1), sigma_v_ekf_wn*1000);
hold on;
h8 = plot(t-t(1), sigma_v_ekf_nn*1000, '--', Color = h7.Color);
set(gca, 'YScale', 'log');
ylabel('Velocity $\sigma$ [m/s]');
xlabel('Elapsed Time [s]');
title('EKF Position and Velocity Standard Deviation');
grid on;
saveas(fig, fullfile('output','ekf_wn.png'));
close(fig);

% Q16
fig = figure('Position',[200 200 1600 900]);
h1 = plot(t-t(1), d_r_ekf_wn*1000, 'DisplayName', 'Position Error $\delta
r$');
hold on;
h2 = plot(t-t(1), obs_res_wn*1000, 'DisplayName', 'Observation Residual RMS
$e_{RMS}$');
ylabel('Error [m]');
xlabel('Elapsed Time [s]');
title('EKF Position Error and Observation Residual RMS with Process Noise');
legend('Location', 'best');
grid on;
saveas(fig, fullfile('output','obs_res.png'));
close(fig);
```

EXTRA: LKF

```
[x_est_lkf, P_est_lkf] = lkf(t, x1, P0, sigma_rho, CA_range, PRN_ID, x_GPS);
% No process noise
d_r_lkf = vecnorm(x_est_lkf(1:3, :) - x_ref(1:3, :), 2, 1); % Position error
over time no process noise
d_v_lkf = vecnorm(x_est_lkf(4:6, :) - x_ref(4:6, :), 2, 1); % Velocity error
over time no process noise
sigma_r_lkf = sqrt(squeeze(P_est_lkf(1,1,:) + P_est_lkf(2,2,:) +
P_est_lkf(3,3,:))); % Position 3-sigma no process noise
sigma_v_lkf = sqrt(squeeze(P_est_lkf(4,4,:) + P_est_lkf(5,5,:) +
P_est_lkf(6,6,:))); % Velocity 3-sigma no process noise

fig = figure('Position',[200 200 1600 900]);
yyaxis left
h1 = plot(t-t(1), d_r_lkf*1000);
```

```
hold on;
h2 = plot(t-t(1), d_r_ekf_nn*1000, '--', Color = h1.Color);
ylabel('Position Error [m]');
yyaxis right
h3 = plot(t-t(1), d_v_lkf*1000);
hold on;
h4 = plot(t-t(1), d_v_ekf_nn*1000, '--', Color = h3.Color);
ylabel('Velocity Error [m/s]');
xlabel('Elapsed Time [s]');
title('LKF vs EKF Position and Velocity Error without Process Noise');
% Empty plots to show legend in black
e1 = plot(nan, nan, '-k', 'DisplayName', 'LKF');
e2 = plot(nan, nan, '--k', 'DisplayName', 'EKF');
legend([e1, e2], 'Location', 'best');
grid on;
saveas(fig, fullfile('output','error_lkf.png'));
close(fig);

fig = figure('Position',[200 200 1600 900]);
yyaxis left
h1 = plot(t-t(1), sigma_r_lkf*1000);
hold on;
h2 = plot(t-t(1), sigma_r_ekf_nn*1000, '--', Color = h1.Color);
ylabel('Position $\sigma$ [m]');
yyaxis right
h3 = plot(t-t(1), sigma_v_lkf*1000);
hold on;
h4 = plot(t-t(1), sigma_v_ekf_nn*1000, '--', Color = h3.Color);
ylabel('Velocity $\sigma$ [m/s]');
xlabel('Elapsed Time [s]');
title('LKF vs EKF Position and Velocity Standard Deviation without Process Noise');
% Empty plots to show legend in black
e1 = plot(nan, nan, '-k', 'DisplayName', 'LKF');
e2 = plot(nan, nan, '--k', 'DisplayName', 'EKF');
legend([e1, e2],'Location', 'best');
grid on;
saveas(fig, fullfile('output','sigma_lkf.png'));
close(fig);
```

FUNCTIONS

```
function C = constants
    % Define constants only once (persistent)
    persistent CONST
    if isempty(CONST)
        % Spacecraft and environmental parameters
        CONST.mass_SC =     600; % kg, Spacecraft Mass
        CONST.Area =        1; % m^2, Reference Surface Area
        CONST.C_D =         2.6; % Aerodynamic Drag Coefficient
        CONST.C_20 =         -4.841692151273e-04; % Gravity Field Coefficient
        CONST.rho_atm =     1e-11; % kg/m^3, Atmospheric Density
        % Physical constants
        CONST.c_light =     299792.458; % km/s, Speed of Light
        CONST.omega_Earth = 7.292115e-05; % rad/s, Earth Rotation Rate
        CONST.GM_Earth =    3.986004415e05; % km^3/s^2, Earth's
Gravitational Constant
        CONST.R_Earth =     6378.1366; % km, Earth's Reference Radius
    end
    C = CONST;
end

function [x_est, P_est, obs_res] = ekf(t, x0, P0, sigma_rho, CA_range,
PRN_ID, x_GPS, Q)
    % Extended Kalman Filter
    % Inputs:
    %   t: 1 x n_time vector of time
epochs                                               (s)
    %   x0: initial state estimate (4x1) [rx; ry; rz; vx; vy;
vz]                         (km, km/s)
    %   P0: initial covariance estimate (6x6)
    %   sigma_rho: standard deviation of pseudorange
measurements                         (km)
    %   CA_range: n_time x N_sats matrix of pseudorange
measurements                      (km)
    %   PRN_ID: n_time x N_sats matrix of PRN IDs (0 if not tracked)
    %   x_GPS: 6 x n_time x N_sats array of GPS satellite states [rx; ry;
rz; vx; vy; vz] (km, km/s)
    %   Q: (optional) process noise covariance matrix (6x6). Defaults to
zero matrix.
    % Outputs:
    %   x_est: estimated states over time (6 x n_time)
    %   P_est: estimated covariances over time (6 x 6 x n_time)
    %   obs_res: RMS per epoch of the observation residuals over time

    % Initialize arrays to store estimates
    x_est = zeros(6, length(t));
    P_est = zeros(size(P0, 1), size(P0, 2), length(t));
    obs_res = zeros(length(t), 1);

    % Set default process noise matrix if not provided
    if nargin < 8
        Q = zeros(6, 6);
    end

    x_pred = x0;
    P_pred = P0;
    for k=1:length(t)
        % Update with observations at t(k)
        tracked_sats = PRN_ID(k, :) > 0;
        z_k = CA_range(k, tracked_sats)'; % Available measurements up to
time t(k)
        x_t = squeeze(x_GPS(:, k, tracked_sats)); % GPS satellite states for
tracked satellites
        [h_x_k, H_k] = observation(x_pred, x_t);
        dz_k = z_k - h_x_k;
        R_k = (sigma_rho^2) * eye(sum(tracked_sats));
        % Kalman Gain
        K_k = (P_pred*H_k') / (H_k*P_pred*H_k' + R_k);
        x_est(:, k) = x_pred + K_k * dz_k;
        P_est(:, :, k) = (eye(6) - K_k*H_k) * P_pred;

        % Compute observation residual RMS
        dx_k = K_k * dz_k;
        e_k = dz_k - H_k * dx_k;
        obs_res(k) = sqrt(mean(e_k.^2));

        % Propagation
        if k==length(t)
            % Last epoch, no need to propagate
            break;
        end
        dt = t(k+1) - t(k);
        [x_pred, Phi] = propagation(x_est(:, k), eye(6), dt);
        P_pred = Phi * P_est(:, :, k) * Phi.' + Q;
    end
end

function [x_est, P_est] = lkf(t, x0, P0, sigma_rho, CA_range, PRN_ID, x_GPS,
Q)
    % Linear Kalman Filter (propagate once, then update)
    % Inputs:
    %   t: 1 x n_time vector of time
epochs                                               (s)
    %   x0: initial state estimate (6x1) [rx; ry; rz; vx; vy;
vz]                         (km, km/s)
    %   P0: initial covariance estimate (6x6)
    %   sigma_rho: standard deviation of pseudorange
measurements                         (km)
    %   CA_range: n_time x N_sats matrix of pseudorange
measurements                      (km)
    %   PRN_ID: n_time x N_sats matrix of PRN IDs (0 if not tracked)
    %   x_GPS: 6 x n_time x N_sats array of GPS satellite states [rx; ry;
rz; vx; vy; vz] (km, km/s)
    %   Q: (optional) process noise covariance matrix (6x6). Defaults to
zero matrix.
    % Outputs:
    %   x_est: estimated states over time (6 x n_time)
    %   P_est: estimated covariances over time (6 x 6 x n_time)

    % Initialize arrays to store estimates
    x_est = zeros(6, length(t));
    P_est = zeros(size(P0, 1), size(P0, 2), length(t));

    % Set default process noise matrix if not provided
    if nargin < 8
        Q = zeros(6, 6);
    end

    % Propagate trajectory once using initial state
    x_ref = zeros(6, length(t));
    Phi_ref = cell(length(t), 1);
    x_ref(:, 1) = x0;
    Phi_ref{1} = eye(6);

    for k=1:length(t)-1
        [x_ref(:, k+1), Phi_ref{k+1}] = propagation(x_ref(:, k), Phi_ref{k},
t(k+1) - t(k));
    end

    % Update with measurements using propagated trajectory
    P_pred = P0;
    for k=1:length(t)
        % Update with observations at t(k) using propagated state
        tracked_sats = PRN_ID(k, :) > 0;
        z_k = CA_range(k, tracked_sats)'; % Available measurements at time
t(k)
        x_t = squeeze(x_GPS(:, k, tracked_sats)); % GPS satellite states for
tracked satellites

        % Compute measurement matrix H and predicted measurements h_x using
propagated state
        [h_x_k, H_k] = observation(x_ref(:, k), x_t);

        % Innovation
        dz_k = z_k - h_x_k;
        R_k = (sigma_rho^2) * eye(sum(tracked_sats));

        % Kalman Gain
        K_k = (P_pred*H_k') / (H_k*P_pred*H_k' + R_k);

        % State update
        x_est(:, k) = x_ref(:, k) + K_k * dz_k;

        % Covariance update
        P_est(:, :, k) = (eye(6) - K_k*H_k) * P_pred;

        % Propagate covariance for next measurement (using original
trajectory)
        if k < length(t)
            Phi_kk1 = Phi_ref{k} / (Phi_ref{k+1});
            P_pred = Phi_kk1 * P_est(:, :, k) * Phi_kk1.' + Q;
        end
    end
end

function load_directory_data(dirname, suffix)
    if nargin < 2
        suffix = '';
    end
    files = dir(fullfile(dirname, '*.txt'));
    for i = 1:length(files)
        fname = files(i).name;
        [~, fileName, ~] = fileparts(fname);
        varName = [fileName, suffix];
        data = load(fullfile(dirname, fname));
        assignin('caller', varName, data);
    end
end

function [h_x, H] = observation(x_r, x_t)
    % Observation model for pseudorange measurements
    % Inputs:
    %   x_r: 6x1 vector [rx; ry; rz; vx; vy; vz] of receiver
    %   x_t: 6xN_sats matrix [rx; ry; rz; vx; vy; vz] for each satellite
    % Outputs:
    %   h_x: N_sats x 1 vector of predicted pseudorange measurements
    %   H: N_sats x 6 design matrix
    N_sats = size(x_t, 2);
    h_x = zeros(N_sats, 1);
    H = zeros(N_sats, 6);

    r_t = x_t(1:3, :);
    r_r = x_r(1:3);

    for i = 1:N_sats
        dist = norm(r_t(:, i) - r_r);
        h_x(i) = dist;
        % Design matrix
        H(i, 1:3) = (r_r - r_t(:, i))' / dist;
    end
end

function [x, Phi] = propagation(x0, Phi0, dt)
% Propagate 6x1 state and 6x6 STM over a time step dt (s) using two-body
dynamics
% Inputs:
%   x0: initial 6x1 state [r; v] in km and km/s (ECEF/ECR frame).
%   Phi0: initial 6x6 state transition matrix.
%   dt: propagation interval in seconds (can be negative).
% Outputs:
%   x: propagated state at t0+dt.
%   Phi: propagated STM (6x6), initialized as identity at t0.

    % Augmented initial state: state + vec(I6)
    g0 = [x0(:); Phi0(:)];

    % ODE integration with ode45
    odefun = @(t, g) dgn(t, g);
    tspan = [0, dt];
    opts = odeset('RelTol', 1e-12, 'AbsTol', 1e-12);
    [~, g] = ode45(odefun, tspan, g0, opts);

    g_end = g(end, :).';
    x = g_end(1:6);
    Phi = reshape(g_end(7:end), 6, 6);
end
```

```
function gdot = dgn(~, g)
    r = g(1:3);
    v = g(4:6);
    Phi = reshape(g(7:end), 6, 6);

    C = constants();
    mu = C.GM_Earth; % km^3/s^2

    % Earth rotation skew matrix
    Omega = [ 0             -C.omega_Earth 0;
              C.omega_Earth 0              0;
              0             0              0]; % rad/s

    r_norm = norm(r);
    % Two-body + Earth rotation accelerations
    a_grav = -(mu / r_norm^3) * r;
    a_rot = -(Omega * Omega) * r - 2 * Omega * v;
    a = a_grav + a_rot;

    % State Jacobian F
    F = [zeros(3), eye(3);
         (mu / r_norm^5) * (3 * (r * r.') - (r_norm^2) * eye(3)) - (Omega *
Omega), -2 * Omega];

    Phi_dot = F * Phi;

    gdot = [v; a; Phi_dot(:)];
end
```

```
Q5:
   Propagated state at t2:
   r(t2) = 132.570096 -29.474403 -6887.499013 km
   v(t2) = -1.924598 7.333033 -0.079419 km/s
   STM at t2:
   Phi_22(t2, t1) = 0.9999393259
   Phi_15(t2, t1) = 0.0072918446
   Phi_36(t2, t1) = 10.0004061184
   Phi_52(t2, t1) = -0.0000121358

Q6:
   EKF without process noise:
   Epoch 10: r = (-18.289785, 557.408155, -6867.067024) km
   Epoch 20: r = (-196.928442, 1286.827767, -6766.479380) km
   Epoch 30: r = (-362.643546, 2003.239584, -6583.732102) km
   Epoch 10: v = (-1.844483, 7.329354, 0.590866) km/s
   Epoch 20: v = (-1.724921, 7.243581, 1.421256) km/s
   Epoch 30: v = (-1.586398, 7.068425, 2.234034) km/s
```

Published with MATLAB® R2025a