KALMAN FILTER

Amanda Román Navarro (6304788)

January 11, 2025 Time spent: 23 hours

a) Covariance matrix of the initial state vector and the covariance matrix of the observations

The Covariance Matrix of the initial state vector is

$$P = \begin{bmatrix} 4 & 0 & 0 & 0.14 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0.14 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0.14 \\ 0.14 & 0 & 0 & 0.01 & 0 & 0 \\ 0 & 0.14 & 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0.14 & 0 & 0 & 0.01 \end{bmatrix}$$

where $\sigma^2 = 4\,\mathrm{m}^2$ is the standard deviation for the initial position, $\sigma^2 = 0.01\,\mathrm{m}^2$ is the standard deviation for the initial velocity and 0.14 is the correlation coefficient between x and v_x, y and v_y; and z and v_z.

The Observation Covariance Matrix for n satellites is

$$R = \sigma_\rho \cdot I_n$$

where $\sigma_\rho = 9$ and I_n is an $n \times n$ identity matrix.

b) Integrated State Vector and State Transition Matrix

To integrate the state transition matrix, it is necessary to solve the following differential equation

$$\frac{d}{dt}\begin{bmatrix} \overline{x}(t) \\ \Phi(t,t_k) \end{bmatrix} = \begin{bmatrix} f(t,\overline{x}(t)) \\ \frac{\partial f(t,\overline{x}(t),\overline{p})}{\partial \overline{x}(t)}\Phi(t,t_k) \end{bmatrix}$$

This coupling arises because the state transition matrix depends on the evolution of the state vector. Each derivative can be expressed separately as

$$\frac{d}{dt}\overline{x}(t) = \begin{bmatrix} \overline{v}(t) \\ \overline{a}(t) \end{bmatrix}$$

$$\frac{\partial f(t,\overline{x}(t),\overline{p})}{\partial \overline{x}(t)}\Phi(t,t_k) = \begin{bmatrix} \frac{\partial \overline{v}(t)}{\partial \overline{r}(t)} & \frac{\partial \overline{v}(t)}{\partial \overline{v}(t)} \\ \frac{\partial \overline{a}(t,\overline{x}(t),\overline{p}(t))}{\partial \overline{r}(t)} & \frac{\partial \overline{a}(t,\overline{x}(t),\overline{p}(t))}{\partial \overline{v}(t)} \end{bmatrix} = \begin{bmatrix} 0_{3x3} & I_{3x3} \\ \frac{\mu}{r^5}\left(3\overline{r}\cdot\overline{r}^T - r^2 I_{3x3} - \Omega^2\right) & -2\Omega \end{bmatrix}$$

where the acceleration vector is given by

$$\overline{a}(t,\overline{x}(t),\overline{p}(t)) = \frac{-\mu}{r^5}\overline{r} - \Omega\left(\Omega\cdot\overline{r}(t)\right) + 2\Omega\overline{v}(t)$$

This equation corresponds to a dynamic model where Earth is considered a point mass, ignoring higher-order gravitational effects but accounting for the accelerations due to Earth's rotation.

The integrated state vector is

$$\overline{x}(t) = \begin{bmatrix} \overline{r}(t)[m] \\ \overline{v}(t)[m/s] \end{bmatrix} \quad \rightarrow \quad \overline{x}(t) = \begin{bmatrix} 844746.579 \\ -4170850.435 \\ -5097453.791 \\ -513.212 \\ -6064.085 \\ 4885.441 \end{bmatrix}$$

The integrated state transition matrix is

$$\Phi(t, t_k) = \begin{bmatrix} 0.9999352572 & -0.0000162171 & -0.0000201008 & 9.9997806480 & 0.0072378941 & -0.0000668707 \\ -0.0000161909 & 1.0000109450 & 0.0000981305 & -0.0073462034 & 10.000034210 & 0.0003273839 \\ -0.0000201485 & 0.0000981207 & 1.0000538020 & -0.0000671095 & 0.0003273350 & 10.000178060 \\ -0.0000129506 & -0.0000032477 & -0.0000040051 & 0.9999341879 & 0.0014421264 & -0.0000199978 \\ -0.0000032398 & 0.0000022681 & 0.0000196446 & -0.0014746669 & 1.0000106530 & 0.0000983038 \\ -0.0000040194 & 0.0000196417 & 0.0000106843 & -0.0000200934 & 0.0000982843 & 1.0000530370 \end{bmatrix}$$

This matrix reflects the sensitivity of the state vector to initial conditions.

c) Extended Kalman Filter

Following the Extended Kalman Filter algorithm, the estimated positions and velocities for the epochs 10, 20 and 30 are

Epoch	**x** [m]	**y** [m]	**z** [m]	$\mathbf{v_x}$ [m/s]	$\mathbf{v_y}$ [m/s]	$\mathbf{v_z}$ [m/s]
10	797335.823	-4636973.530	-4684986.718	-670.312	-5579.575	5418.653
20	721171.929	-5161831.965	-4112450.993	-849.926	-4903.743	6018.595
30	628111.244	-5615588.073	-3483899.906	-1007.739	-4158.625	6536.947

And the position and velocity errors between the estimated and precise position and velocity are

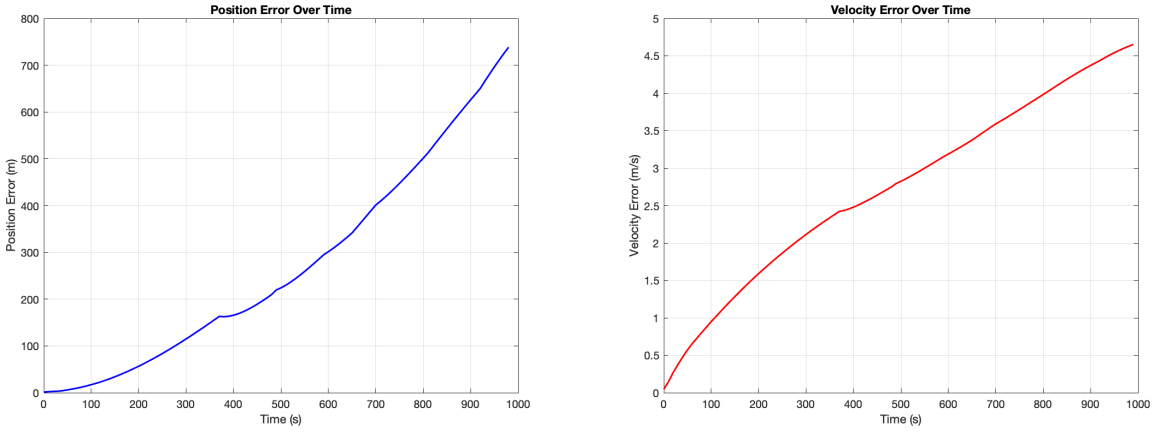

Figure 1: Position and Velocity errors over time

Both velocity and position estimates show a steady increase of the error with respect to the precise position and velocity.

d) Standard deviation of the estimated position and velocity

The standard deviation of the estimated position and velocity are

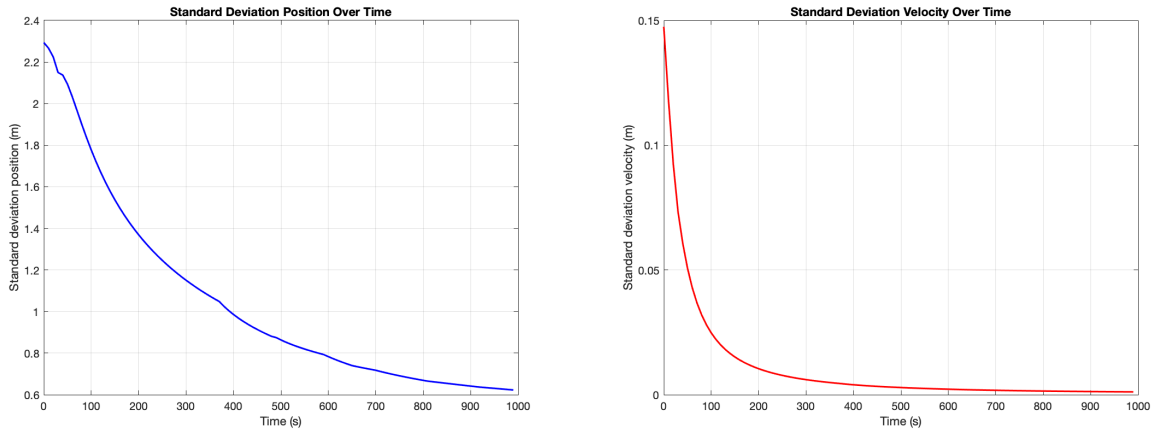

Figure 2: Standard Deviation for Position and Velocity over time

The standard deviations represent the Kalman Filter's "confidence" in the estimated position and velocity states over time. Initially, both standard deviations start at relatively high values because the filter begins with limited observational data and relies heavily on its initial predictions about state uncertainty. As the Kalman Filter processes more observations through the epochs, it reduces these uncertainties with repeated update and propagation steps, which is reflected in the state covariance matrix.

For σ_r, the decrease happens steadily and gradually flattens out over time. This shows the Kalman Filter's increasing confidence in the position estimates as the observational data increases the accuracy on the state prediction. In contrast, σ_v shows a much sharper initial decline, quickly reaching an minimum value. This rapid stabilization of the velocity is due to the fact that the velocity components are not directly observed but instead obtained through the state integration and Kalman gain. This causes the filter to heavily correct the position states through both predictions and observations, while the velocity states are obtained from the dynamic model.

The position error δ_r increases significantly over time, and this growth is highly driven by the accumulation of numerical integration errors and inaccuracies in the initial state. Propagation errors, especially in position, are magnified because position depends on the integral of velocity, meaning small errors in velocity estimates accumulate over time and lead to larger deviations in position. The velocity error δ_v , on the other hand, grows more linearly, which suggest that the filter's velocity state updates are not sufficiently constrained, as this only relies on the dynamic model and has no correction by the observations.

Despite the decreasing standard deviations σ_r and σ_v the increasing errors δ_r and δ_v suggest that the Kalman Filter's dynamic model does not perfectly represent the true system behavior or is overly reliant on state propagation between measurements. This leads to accumulated inaccuracies, especially in the position estimates.

e) Linear Kalman Filter

If a Linear Kalman Filter was used to calculate position and velocity instead of the Extended Kalman Filter, the position errors would be larger than those observed in task c with the Extended Kalman Filter. This difference would be caused by how the two filters handle non-linearities in the system dynamics and measurement models. The Linear Kalman Filter assumes both the state transition and measurement models are linear. However, orbital dynamics and measurement such as pseudorange observations from GPS satellites, are nonlinear. In the Linear Kalman Filter, the state transition matrix Φ and observation matrix H are linearized around a fixed reference point, often the initial state estimate. Over time numerical integration errors, system uncertainties, and poor initial assumptions would cause the filter's predictions to deviate highly from the true state. The fixed linearization wouldn't adapt to the current state trajectory, making

the filter less accurate and inadequate for this case.

On the other hand, the Extended Kalman Filter is adecuate for non-linear systems, as this re-linearizes both the state transition matrix and observation model at each time step using the previous state estimate. This constant re-linearization ensures that the approximations remain valid locally, minimizing linearization errors and maintaining a closer approximation to the true trajectory. Additionally, it integrates the state transition matrix and state vector simultaneously, which shows better the evolution of system dynamics between epochs. Moreover, the Extended Kalman Filter incorporates the process noise (Q), which prevents the filter from becoming overly confident in its predictions, deviating the estimates from the real state.

f) Process noise

In the Kalman Filter, process noise is introduced to address uncertainties and imperfections in the system's dynamic model. It accounts for unmodeled dynamics, external disturbances, and numerical approximations that accumulate during state propagation. Without accounting for process noise, the Kalman Filter may become overconfident in its predictions, leading to the filter ignoring observational updates and relying excessively on the model's propagation step, leading to a continuous increase of the error. These uncertainties are represented by the process noise covariance matrix, where the parameters σ_a and σ_b quantify the noise contributions to the dynamic model for position and velocity.

To analyze the effect of process noise on the filter's performance, various combinations of σ_a and σ_b have been tested. These parameters were selected by observing the initial covariance matrix of the initial state vector and evaluating their impact on both position and velocity errors, as well as the standard deviations of the estimated states.

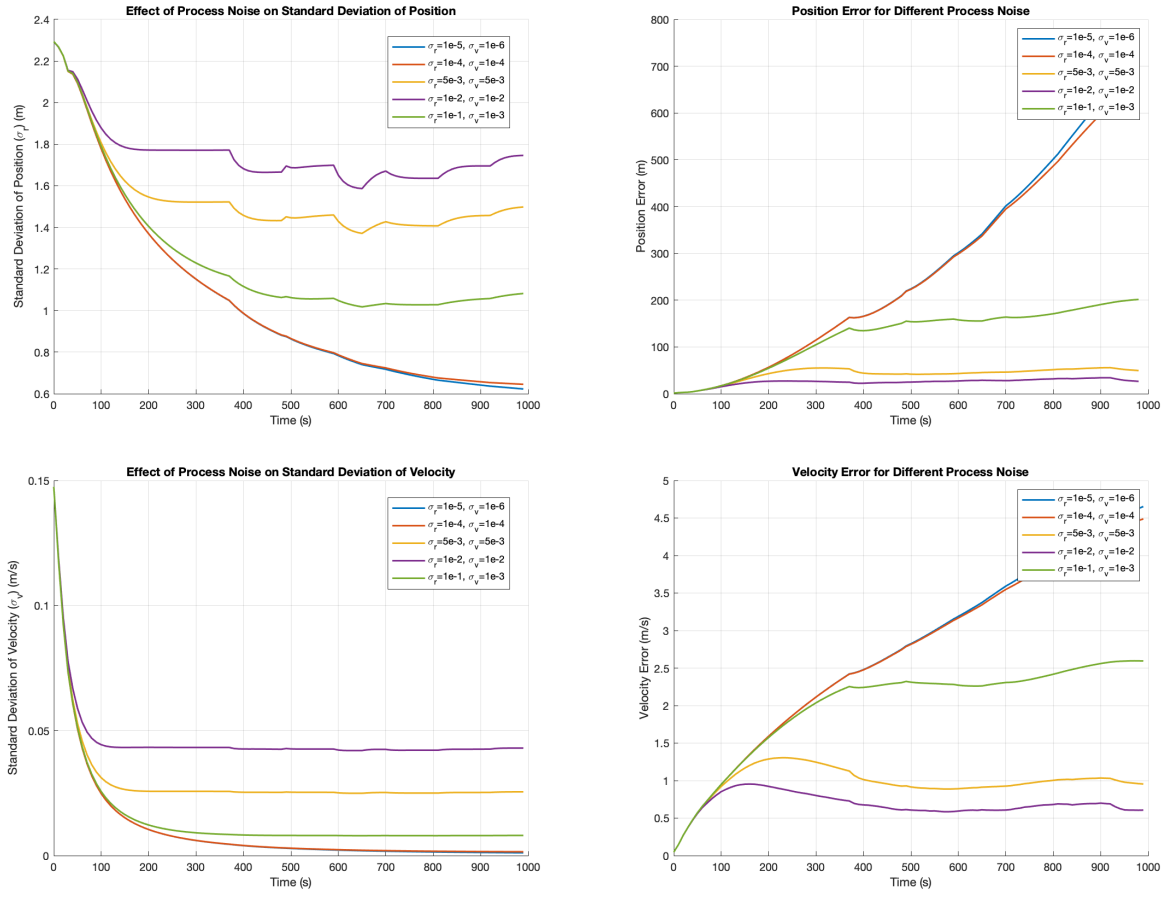

Figure 3: Effect of Noise Covariance Matrix on Position and Velocity errors and standard deviations

From the analysis shown in the figure, the optimal values for the process noise covariance matrix were determined to be $\sigma_a = 5 * 10^{-3}$ and $\sigma_b = 5 * 10^{-3}$ (represented by the yellow line in the plots). These values balance the dynamic model's and the filter's responsiveness to observational updates. Higher process noise values resulted in the filter relying excessively on observational data, leading to underestimating the importance of the dynamic model and almost neglecting the integration step performed by the filter. The remaining errors should be diminished by a more detailed dynamical model, not by undermining the filter.

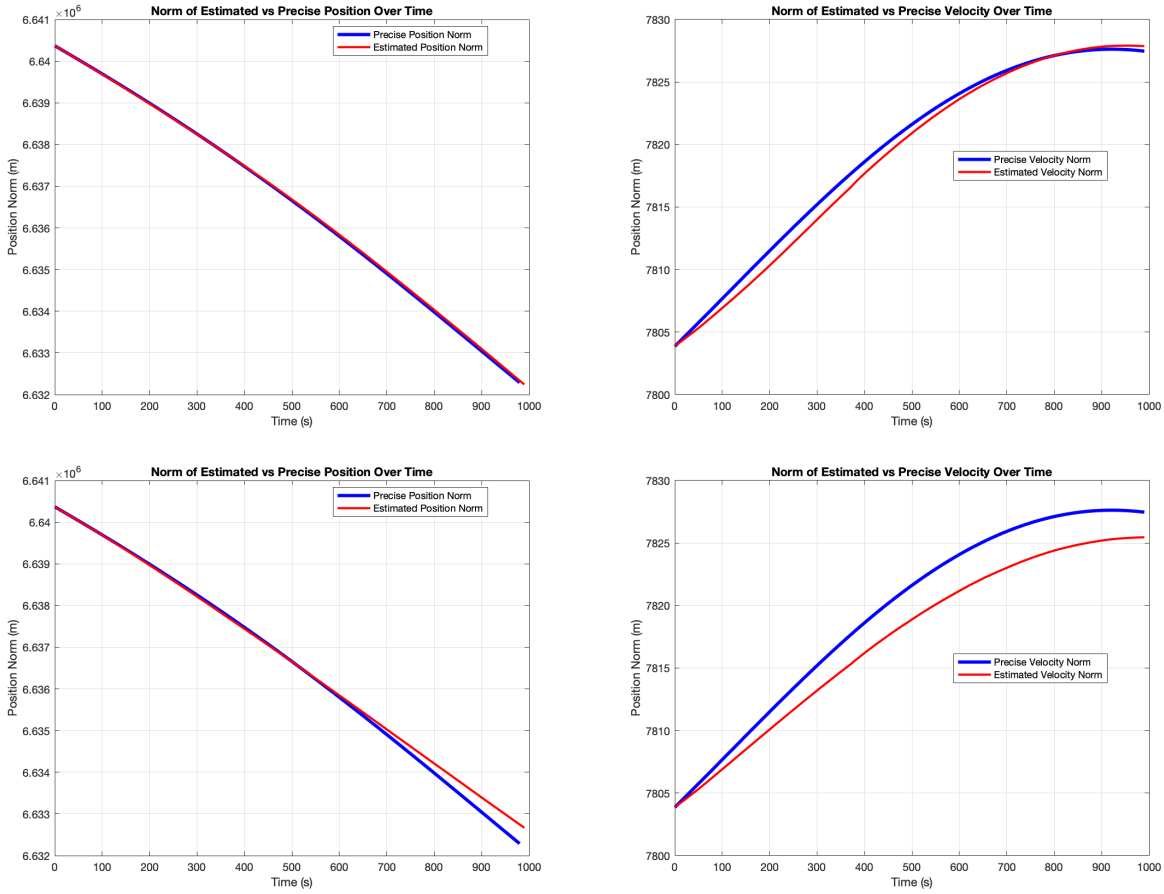

Figure 4: Estimated vs Precise Positions and Velocities

In the plots, the impact of accounting for process noise is visible. The top plots represent results with the process noise covariance matrix included, while the bottom plots show results without process noise. With an appropriate tuned noise matrix, both position and velocity errors are minimized. This demonstrates the importance of including process noise in maintaining filter stability and improving estimation accuracy.

g) RMS

The RMS residuals in the context of a Kalman Filter represent how well the filter's predicted measurements align with the actual observations after applying the correction step. They provide an indication of the filter's ability to calculate model predictions with observed data.

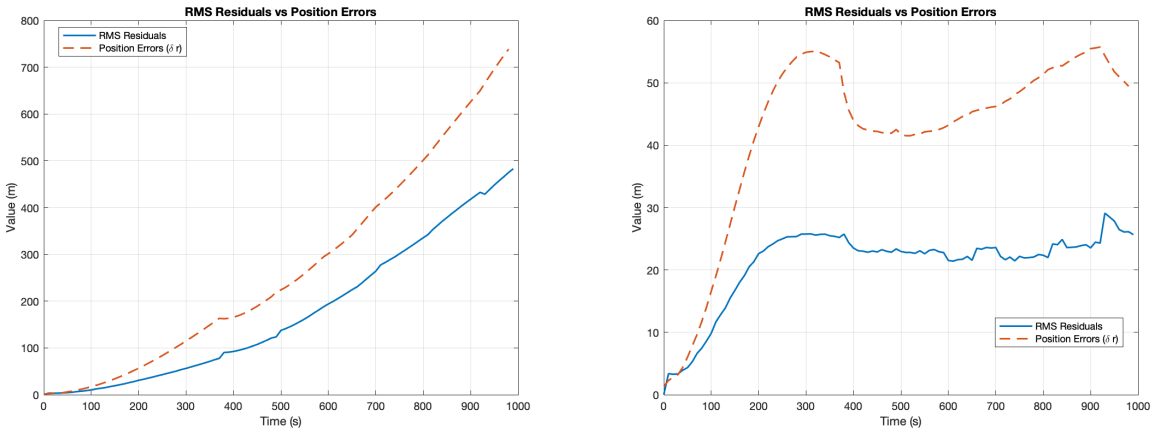

Figure 5: RMS with and without Noise Covariance Matrix

In the left plot, where the process noise covariance matrix Q is not included, the RMS residuals remain relatively stable at low values, while the position errors (δ_r) grow significantly over time. This indicates that the filter is not fitting the observations well over extended periods, as it doesn't account for unmodeled dynamics, external disturbances, and numerical propagation errors. As a result, the position estimates drift over time.

In contrast, the right plot incorporates the process noise covariance matrix Q. Here, the RMS residuals stabilize at low values, and the position errors grow at a slower rate. Including Q introduces an additional refinement to avoid the filter to go to overestimate the integrated state, balancing the weight between predicted states and observational updates more effectively. However, while its introduction reduces the growth rate of position errors, it cannot entirely eliminate propagation errors, which could be caused by unmodeled forces in the dynamic model.

h) More detailed dynamic model

To implement a more detailed dynamic model, the system dynamics have to be expanded to include higher-order gravitational terms, such as spherical harmonics accounting for Earth's flattening, third-body perturbations, atmospheric drag using density models, radiation pressure, and relativistic effects. The state transition matrix (Φ) and sensitivity matrix (S) must be propagated alongside the state vector as follows.

$$\frac{d\Phi(t,t_k)}{dt} = \frac{\partial f(t,x(t),p)}{\partial x(t)} \cdot \Phi(t,t_k), \qquad \frac{dS(t)}{dt} = \frac{\partial f(t,x(t),p)}{\partial x(t)} \cdot S(t) + \frac{\partial f(t,x(t),p)}{\partial p}$$

where $f(t,x(t),p)$ represents the equations of motion including all forces and p denotes the model parameters. These equations must be solved as a coupled system of differential equations alongside the state vector using advanced numerical methods, such as Runge-Kutta or adaptive step-size integrators.

In addition, the process noise covariance matrix (Q_k) must be defined to account for uncertainties in the dynamic model. This matrix can be modeled based on variances in acceleration (σ_a^2) and velocity (σ_b^2) as previously done. To implement these in the existing code, the equations of motion must be updated to include all relevant forces, and the Jacobian matrix for the state vector derivatives must be recalculated. The numerical integration routine should be modified to propagate the state vector, state transition matrix, and sensitivity matrix.

Despite these implementations, some limitations and inaccuracies will persist. Atmospheric drag and radiation pressure are complicated to model accurately due to their dependency on factors such as solar activity, spacecraft surface, and atmospheric density fluctuations. Initial state uncertainties will continue to propagate through the system, observational noise and biases will still influence the estimation, and numerical integration errors will accumulate over time. Additionally, time synchronization errors and unmodeled small-scale forces will introduce residual inaccuracies.

Appendix - Matlab code

```
%% CONSTANTS
clear, clc

%% LOAD DATA

% Load Data
load('data/t.txt');         % epochs - s
load('data/CA_range.txt'); % pseudorange observations - km
load('data/PRN_ID.txt');    % PRN IDs
load('data/rx_gps.txt');    % Satellite positions - km
load('data/ry_gps.txt');
load('data/rz_gps.txt');
load('data/vx_gps.txt');    % Satellite velocities - km/s
load('data/vy_gps.txt');
load('data/vz_gps.txt');
load('data/rx.txt');        % Receiver precise positions - km
load('data/ry.txt');
load('data/rz.txt');
load('data/vx.txt');        % Receiver precise velocities - km/s
load('data/vy.txt');
load('data/vz.txt');

% Convert all units to meters and m/s
CA_range = CA_range * 1e3;
rx_gps = rx_gps * 1e3; ry_gps = ry_gps * 1e3; rz_gps = rz_gps * 1e3;
vx_gps = vx_gps * 1e3; vy_gps = vy_gps * 1e3; vz_gps = vz_gps * 1e3;
rx = rx * 1e3; ry = ry * 1e3; rz = rz * 1e3;
vx = vx * 1e3; vy = vy * 1e3; vz = vz * 1e3;

%% B)

% Define vectors and matrix
x0_pred = [rx(1); ry(1); rz(1); vx(1); vy(1); vz(1)]; % Initial state vector

phi0 = eye(6); % Initial state transition matrix

phi0 = reshape(phi0, 36, 1); % Reshape transition matrix

g = [x0_pred; phi0]; % Equations vector

% Time to integrate
time = [t(1), t(2)];

% Numerical integration
[t_f,y_f] = ode45(@(t,y) dydt(t,y), time, g);

% Extract state and transition matrix
xf = y_f(end, 1:6); % integrated position and velocity
phi_f = reshape(y_f(end, 7:end), 6, 6); % integrated state transition matrix


disp('Final State Vector (Position and Velocity):');
fprintf('Position (m): %.6f %.6f %.6f\n', xf(1:3));
fprintf('Velocity (m/s): %.6f %.6f %.6f\n', xf(4:6));

disp('Final State Transition Matrix:');
disp(vpa(phi_f, 10));

%% FUNCTION DEFINITION
function dy = dydt(t,y)
    % Constants
    mu = 398600.4415e9; % Gravitational parameter (m^3/s^2)
    omega = 7.292115e-5; % Earth's rotation rate (rad/s)

```

```
    % Extract state
    r = y(1:3); % Position
    v = y(4:6); % Velocity
    phi = reshape(y(7:end), 6, 6); % State transition matrix

    % Omega matrix
    Omega = [0 -omega 0; omega 0 0; 0 0 0];

    % Acceleration Components
    a_gravity = - mu / norm(r)^3 * r;
    a_centrifugal = - Omega * Omega * r;
    a_coriolis = - 2 * Omega * v;

    % Total Acceleration
    a = a_gravity + a_centrifugal + a_coriolis;

    % Jacobian Matrix (State Transition)
    F21 = (mu / norm(r)^5) * (3 * (r * r') - norm(r)^2 * eye(3)) - Omega^2; % Partial wrt ...
        position
    F22 = -2 * Omega; % Partial wrt velocity

    F = [zeros(3), eye(3);
         F21 + Omega * Omega, F22];

    % State Transition Derivative
    dphi = F * phi;

    % State Derivative
    dy = [v; a; reshape(dphi, 36, 1)];
end

%% C)
%% INITIALIZATION

% Preallocate results
x_est = zeros(length(t), 6);
position_errors = zeros(length(t), 1);
velocity_errors = zeros(length(t), 1);
sigma_r = zeros(length(t), 1);
sigma_v = zeros(length(t), 1);
RMS = zeros(length(t), 1);

% Initial State Vector (Position and Velocity from Precise Orbit)
x0 = [rx(1); ry(1); rz(1); vx(1); vy(1); vz(1)];

% Initial State Covariance Matrix
sig_r = 2;  % Position std [m]
sig_v = 0.1; % Velocity std [m/s]

P0 = [sig_r^2, 0, 0, 0.7 * sig_r * sig_v, 0, 0;
      0, sig_r^2, 0, 0, 0.7 * sig_r * sig_v, 0;
      0, 0, sig_r^2, 0, 0, 0.7 * sig_r * sig_v;
      0.7 * sig_r * sig_v, 0, 0, sig_v^2, 0, 0;
      0, 0.7 * sig_r * sig_v, 0, 0, sig_v^2, 0;
      0, 0, 0.7 * sig_r * sig_v, 0, 0, sig_v^2];

% Process Noise Covariance Matrix (According to Slides)
%sigma_a = 0;
%sigma_b = 0;

sigma_a = 1;
sigma_b = 1e-1;

Q = [sigma_a^2 * eye(3), zeros(3);
     zeros(3), sigma_b^2 * eye(3)];

% Initial observation update
epoch = 1;
```

```
132
133 valid_PRN = find(PRN_ID(epoch, :) > 0);
134 n_sats = length(valid_PRN);
135
136 % Observation Covariance Matrix
137 sigma_obs = 3; % Pseudorange std [m]
138 R0 = sigma_obs^2 * eye(n_sats);
139
140 %% INITIAL UPDATE
141
142 % Observation Matrix H
143 H0 = zeros(n_sats, 6);
144 xt = rx_gps(epoch, valid_PRN)';
145 yt = ry_gps(epoch, valid_PRN)';
146 zt = rz_gps(epoch, valid_PRN)';
147
148 for i = 1:n_sats
149     dx = x0(1) - xt(i);
150     dy = x0(2) - yt(i);
151     dz = x0(3) - zt(i);
152     r = sqrt(dx^2 + dy^2 + dz^2);
153     H0(i, :) = [dx / r, dy / r, dz / r, 0, 0, 0];
154 end
155
156 % Predicted Observation
157 z0_obs = CA_range(epoch, valid_PRN)';
158 z0_pred = sqrt((x0(1) - xt).^2 + (x0(2) - yt).^2 + (x0(3) - zt).^2);
159 Δ_z0 = z0_obs - z0_pred;
160
161 % Kalman Gain
162 K0 = P0 * H0.' / (H0 * P0 * H0.' + R0);
163
164 % State Update
165 x0_upd = x0 + K0 * Δ_z0;
166 P0_upd = (eye(6) - K0 * H0) * P0;
167
168 % Store Initial Results
169 x_est(epoch, :) = x0_upd';
170 sigma_r(epoch) = sqrt(P0_upd(1,1) + P0_upd(2,2) + P0_upd(3,3));
171 sigma_v(epoch) = sqrt(P0_upd(4,4) + P0_upd(5,5) + P0_upd(6,6));
172
173 % Calculate Position and Velocity Errors
174 position_errors(epoch) = norm(x0_upd(1:3) - [rx(epoch); ry(epoch); rz(epoch)]);
175 velocity_errors(epoch) = norm(x0_upd(4:6) - [vx(epoch); vy(epoch); vz(epoch)]);
176
177 %% PROPAGATE TO EPOCH t1
178 Phi0 = eye(6);
179
180 g = [x0_upd; reshape(Phi0, 36, 1)];
181 [¬, y_prop] = ode45(@(t, y) dydt(t, y), [t(1), t(2)], g);
182
183 x_prop = y_prop(end, 1:6)';
184 phi_prop = reshape(y_prop(end, 7:end), 6, 6);
185 P_prop = phi_prop * P0_upd * phi_prop' + Q;
186
187 %% MAIN EKF LOOP
188 k = 2;
189 while k≤length(t)
190     % Update Step
191     valid_PRN = find(PRN_ID(k, :) > 0);
192     n_sats = length(valid_PRN);
193
194     xt = rx_gps(k, valid_PRN)';
195     yt = ry_gps(k, valid_PRN)';
196     zt = rz_gps(k, valid_PRN)';
197     z = CA_range(k, valid_PRN)';
198
199     % Predicted Observation
```

```
    range = sqrt((x_prop(1) - xt).^2 + (x_prop(2) - yt).^2 + (x_prop(3) - zt).^2);
    Δ_z = z - range;

    % Observation Matrix H
    H = zeros(n_sats, 6);
    for i = 1:n_sats
        dx = x_prop(1) - xt(i);
        dy = x_prop(2) - yt(i);
        dz = x_prop(3) - zt(i);
        r = sqrt(dx^2 + dy^2 + dz^2);
        H(i, :) = [dx / r, dy / r, dz / r, 0, 0, 0];
    end

    % Observation Covariance
    R = sigma_obs^2 * eye(n_sats);

    % Kalman Gain
    K = P_prop * H.' / (H * P_prop * H.' + R);

    % State Update
    x_upd = x_prop + K * Δ_z;
    P_upd = (eye(6) - K * H) * P_prop;

    % g) Calculate the RMS
    residuals = Δ_z - H * K * Δ_z;
    RMS(k) = sqrt(mean(residuals.^2));

    % Store results
    x_est(k, :) = x_upd';
    sigma_r(k) = sqrt(P_upd(1,1) + P_upd(2,2) + P_upd(3,3));
    sigma_v(k) = sqrt(P_upd(4,4) + P_upd(5,5) + P_upd(6,6));

    % Store Errors
    position_errors(k) = norm([rx(k); ry(k); rz(k)] - x_upd(1:3));
    velocity_errors(k) = norm([vx(k); vy(k); vz(k)] - x_upd(4:6));

    % Propagation Step
    if k < length(t)
        phi = eye(6); % Restart Linearization
        g = [x_upd; reshape(phi, 36, 1)];
        options = odeset('RelTol', 1e-9, 'AbsTol', 1e-12);
        [¬, y_prop] = ode45(@(t, y) dydt(t, y), [t(k), t(k+1)], g, options);

        x_prop = y_prop(end, 1:6)';
        phi_prop = reshape(y_prop(end, 7:end), 6, 6);
        P_prop = phi_prop * P_upd * phi_prop' + Q;

    end

    k = k+1;
end

%% Epochs 10, 20, 30
selected_epochs = [10, 20, 30];
results = x_est(selected_epochs, :);
fprintf('Estimated State Vector at Selected Epochs :\n');
for i = 1:length(selected_epochs)
    fprintf('Epoch %d:\n', selected_epochs(i));
    fprintf('Position (m): %.3f %.3f %.3f\n', results(i,1:3) );
    fprintf('Velocity (m/s): %.3f %.3f %.3f\n', results(i,4:6));
end

%% PLOT POSITION AND VELOCITY ERRORS

% Plot errors position
figure;
plot(t-t(1), position_errors, 'b', 'LineWidth', 1.5);
xlabel('Time (s)');
```

```
ylabel('Position Error (m)');
title('Position Error Over Time');
grid on;
% Plot errors velocity
figure;
plot(t-t(1), velocity_errors, 'r', 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('Velocity Error (m/s)');
title('Velocity Error Over Time');
grid on;


% Plot standard deviaiton position
figure;
plot(t-t(1), sigma_r, 'b', 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('Standard deviation position (m)');
title('Standard Deviation Position Over Time');
grid on;

% Plot standard deviaiton velocity
figure;
plot(t-t(1), sigma_v, 'r', 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('Standard deviation velocity (m)');
title('Standard Deviation Velocity Over Time');
grid on;

norm_estimated = sqrt(x_est(:,1).^2 + x_est(:,2).^2 + x_est(:,3).^2);
norm_precise = sqrt(rx.^2 + ry.^2 + rz.^2);

% Plot Norms Over Time
figure;
plot(t - t(1), norm_precise, 'b', 'LineWidth', 3, 'DisplayName', 'Precise Position Norm');
hold on;
plot(t - t(1), norm_estimated, 'r', 'LineWidth', 2, 'DisplayName', 'Estimated Position Norm');
xlabel('Time (s)');
ylabel('Position Norm (m)');
title('Norm of Estimated vs Precise Position Over Time');
legend('Location', 'Best');
grid on;

norm_estimated_v = sqrt(x_est(:,4).^2 + x_est(:,5).^2 + x_est(:,6).^2);
norm_precise_v = sqrt(vx.^2 + vy.^2 + vz.^2);

% Plot Norms Over Time
figure;
plot(t - t(1), norm_precise_v, 'b', 'LineWidth', 3, 'DisplayName', 'Precise Velocity Norm');
hold on;
plot(t - t(1), norm_estimated_v, 'r', 'LineWidth', 2, 'DisplayName', 'Estimated Velocity ...
    Norm');
xlabel('Time (s)');
ylabel('Position Norm (m)');
title('Norm of Estimated vs Precise Velocity Over Time');
legend('Location', 'Best');
grid on;

%% Plot RMS Residuals vs Position Errors
figure;
plot(t-t(1), RMS, 'LineWidth', 1.5, 'DisplayName', 'RMS Residuals');
hold on;
plot(t-t(1), position_errors, '--', 'LineWidth', 1.5, 'DisplayName', 'Position Errors (\Δ ...
    r)');
xlabel('Time (s)');
ylabel('Value (m)');
title('RMS Residuals vs Position Errors');
legend('Location', 'best');
grid on;
```