Sunay Neelimathara

AERSP 450 - Professor Eapen

Homework #4 Report

The purpose of this homework was to analyze and convert given sensor data to understand the nature of the data and the propagated error across different methods of calculation for DCMs and quaternions. The given data was in the form of a CSV (comma separated variable) file, which I opened using readtable() command with the filename. I then separated the angular velocities into x, y, and z directions and used the given formula to get the seconds (time) for use in the rest of the problems. The problem statement gave 3 angles, a yaw of 30°, a pitch of 70°, and a roll of 20°. The yaw-pitch-roll angles translate to a 3-2-1 Euler angle sequence, as roll is angle 1, pitch is angle 2, and yaw is angle 3. To convert the yaw-pitch-roll angles to a DCM, I utilized the Euler angle reference from the textbook to convert to a DCM:

$$\begin{bmatrix} c\theta_2 c\theta_1 & c\theta_2 s\theta_1 & -s\theta_2 \\ s\theta_3 s\theta_2 c\theta_1 - c\theta_3 s\theta_1 & s\theta_3 s\theta_2 s\theta_1 + c\theta_3 c\theta_1 & s\theta_3 c\theta_2 \\ c\theta_3 s\theta_2 c\theta_1 + s\theta_3 s\theta_1 & c\theta_3 s\theta_2 s\theta_1 - s\theta_3 c\theta_1 & c\theta_3 c\theta_2 \end{bmatrix} \quad (1)$$

Using this in MATLAB, the output DCM is shown in Figure 1:

```
DISPLAYING DCM:

 The DCM for the yaw-pitch-roll angles (3-2-1 euler angle sequence) is as follows:

    0.2962    0.1710   -0.9397
   -0.1915    0.9745    0.1170
    0.9357    0.1453    0.3214


DCM VERIFICATION:

||DCM|| =      1

DCM is valid.
```

Figure 1. DCM output from MATLAB

The DCM verification comes from taking the determinant of the DCM with det() in MATLAB. A determinant of 1 means that the DCM is a valid DCM.

To convert the Euler Angles to quaternions, we use the DCM from Figure 1 to convert to quaternions using the following equations:

$$\beta_0 = \pm\frac{1}{2}\sqrt{C_{11} + C_{22} + C_{33} + 1}$$

$$\beta_1 = \frac{C_{23} - C_{32}}{4\beta_0}$$

$$\beta_2 = \frac{C_{31} - C_{13}}{4\beta_0}$$

$$\beta_3 = \frac{C_{12} - C_{21}}{4\beta_0} \tag{2}$$

I used these equations in MATLAB to solve for the quaternions, and printed Figure 2 as the output:

```
The quaternions for the yaw-pitch-roll angles (3-2-1 euler angle sequence) are as follows:

    Beta0 =     0.8050

    Beta1 =    -0.0088

    Beta2 =     0.5824

    Beta3 =     0.1126
```

Figure 2. Output quaternions for the 3-2-1 Euler angle sequence

The problem statement also asked for the plot of angular velocities vs. time, which were all given in the provided file. Plotting these yields Figure 3:

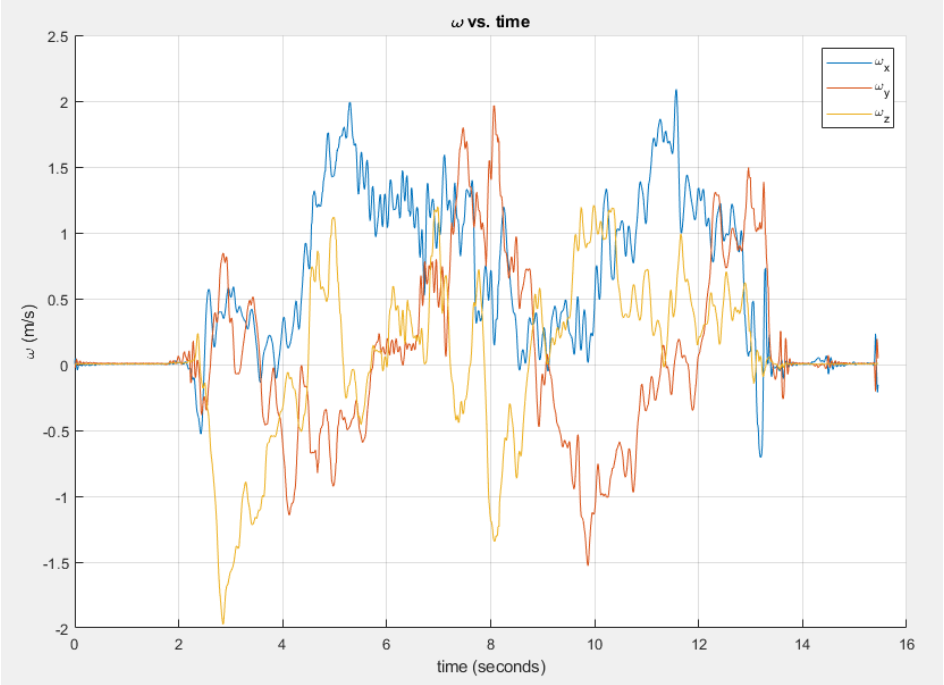

Figure 3. Plot of Angular Velocities vs. Time graph

The angular velocities in the above graph are split into each component in the 3D plane: x, y, and z. This graph shows how they vary across the sensor data given to us over the time span which the sensor was active.

The frequency of the sensor output is calculated as the difference between times in the sensor output (converted to seconds), and then averaging that over the time span given in the CSV file, in MATLAB. After that, taking its reciprocal will provide the sensor frequency, which was output in Figure 4:

```
Sensor Frequency: 99.87 Hz
```

Figure 4. Sensor frequency output from MATLAB

Upon analysis of the data produced by the difference between times of the sensor output, we see that the sensor outputs aren't at equal time intervals. They are not perfectly equal time intervals, and sometimes are higher or lower than 0.01 seconds, though they generally fluctuate around that value.

For Problem 2 in the problem set, we were tasked to propagate the attitude motion using DCMs. The first step required using numerical propagation on the DCM from time t_k to t_{k+1} (k = 0, …) using the equations of motion we found in class. We were also to assume a zero-order-hold for the angular velocities, which means that we assume the discrete time signal is continuous. The equation we use for this problem is:

$$\dot{C}_{BN} = -[\widetilde{\omega}]C_{BN} \tag{3}$$

The $[\widetilde{\omega}]$ is a skew-symmetric omega matrix:

$$[\widetilde{\omega}] = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{4}$$

Using ode45, I found the DCM using the omega matrix and the predetermined C_{BN} matrix. I had to reshape the DCM into a flat vector, then after the ode45, reshape the output to a 3x3 matrix again. Afterwards, I had an output of t and an output of C, which was reshaped to a matrix. This became the numerical DCM matrix.

For the next section, we were to use:

$$C_{BN}(t_{k+1}) = expm(-[\widetilde{\omega}]\Delta t_k)\, C_{BN}(t_k) \tag{5}$$

This produces an analytical result for propagating the DCM across a delta time, which was equal to t_{k+1} - t_k. I solved this in the code to get the analytical DCM matrix.

Our given error equation was:

$$error(t_k) = C_{BN_{(\text{numerical})}}(t_k) C^T_{BN_{(\text{analytical})}}(t_k) - I_{3\times3} \tag{6}$$

We used equation 6 to produce a plot of time history of the error DCM, which is shown below in Figure 5.

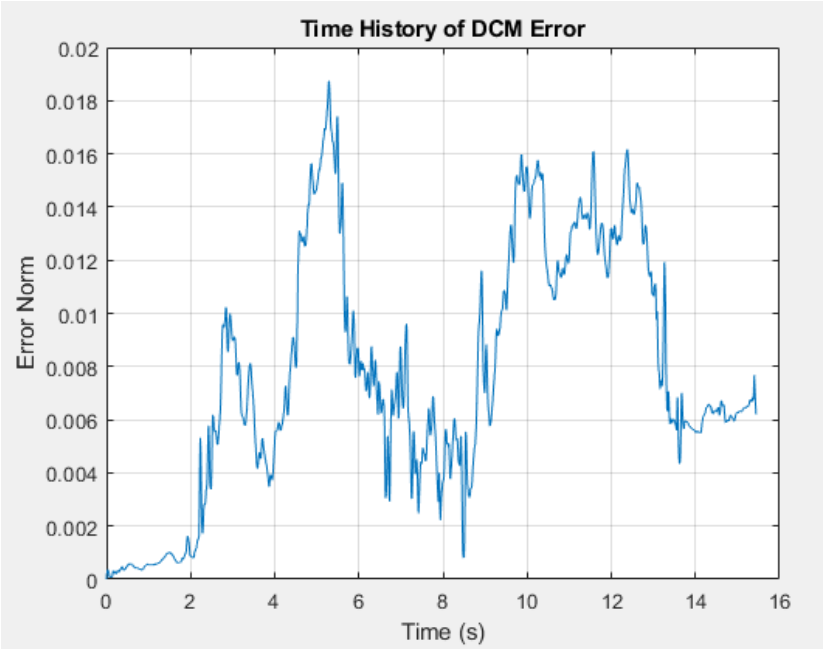

Figure 5. Time history of DCM error graph

The graph shows that error seems to trend upward over time and fluctuates up and down, starting at zero, and approaching it at the end, as well.

Next, we were tasked to plot the yaw-pitch-roll angles as a function of time which are shown in Figure 6 below.

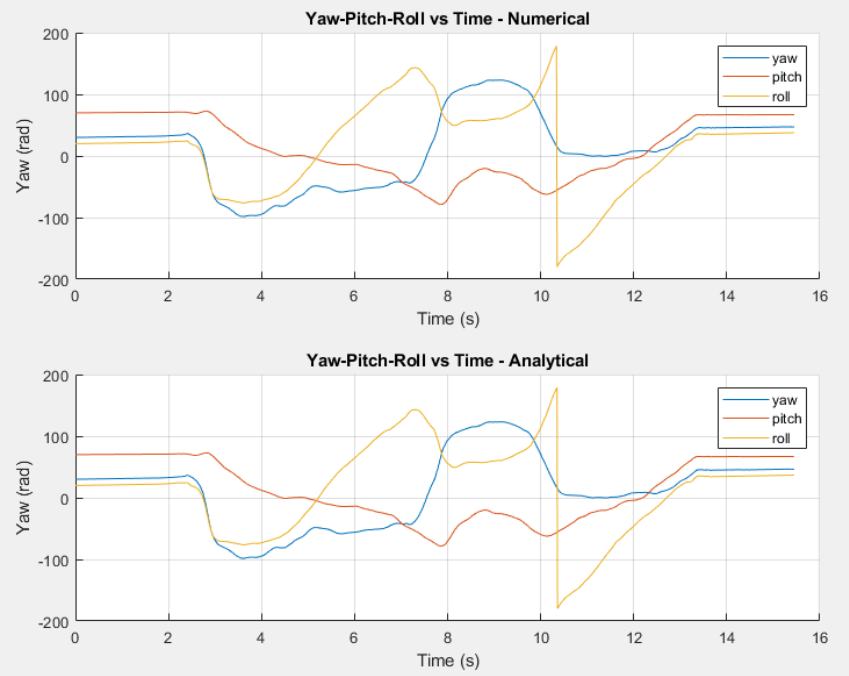

Figure 6. Yaw-Pitch-Roll vs. Time Graphs

These both follow the same trends, which showcases the precision of both methods of determining our DCMs.

For Problem 3 of this assignment, we were tasked with propagating the attitude motion using quaternions. First, we had to analytically propagate the quaternions using the following equation:

$$\bar{\beta}(t_{k+1}) = \Phi(t_k, t_{k+1})\bar{\beta}(t_k), \quad \Phi(t_k, t_{k+1}) = e^{\frac{1}{2}B(\bar{\omega}_{\mathcal{B}/\mathcal{N}})\Delta t}, \quad B(\bar{\omega}_{\mathcal{B}/\mathcal{N}}) = \begin{bmatrix} 0 & -\omega_1 & -\omega_2 & -\omega_3 \\ \omega_1 & 0 & \omega_3 & -\omega_2 \\ \omega_2 & -\omega_3 & 0 & \omega_1 \\ \omega_3 & \omega_2 & -\omega_1 & 0 \end{bmatrix}$$

By using this in the code with a for loop across the time given in the CSV file, I was able to produce a quaternion matrix corresponding to the times given in the file. Next, we were to plot these values across the time span and show that the quaternion constraint was satisfied. The quaternion constraint is that the magnitude of the quaternion vector is equal to 1, so no individual quaternion will cross that value of 1. As shown in Figure 7, no quaternion crosses that line.

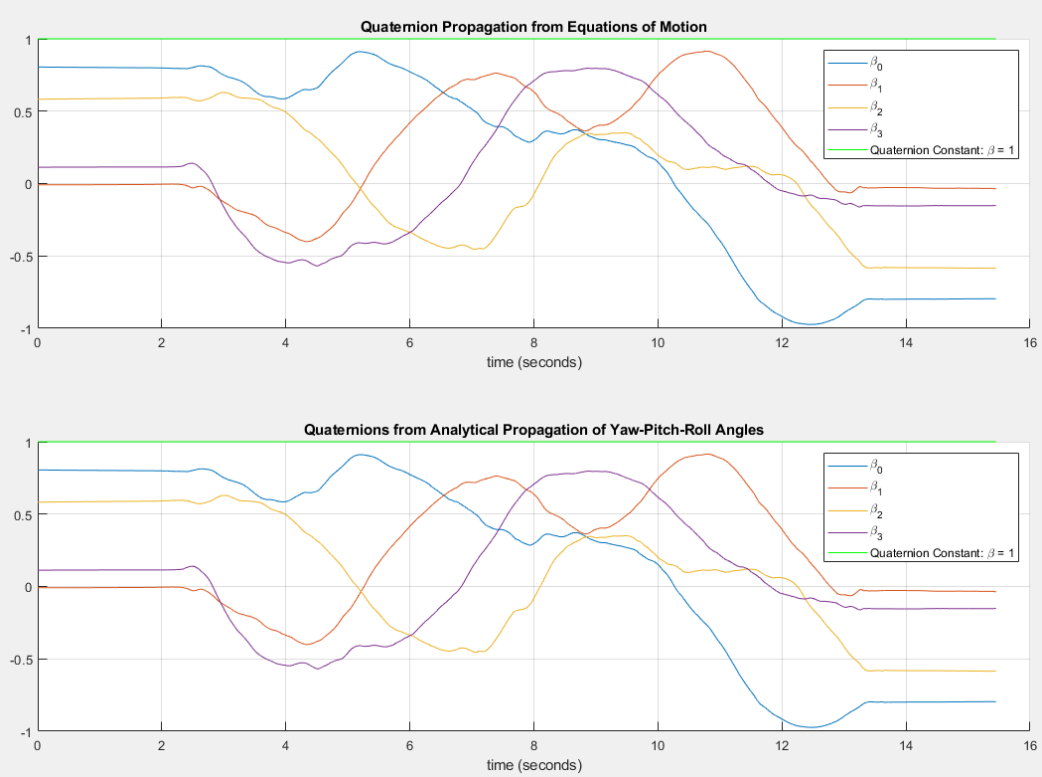

Figure 7. Quaternion Evolution vs. Time for both Analytical and Equations of Motion

Next, we were to convert our yaw-pitch-roll angles from earlier graph into quaternions, which I also graphed in Figure 7. To do so, I used the textbook to find an equation relating the DCMs to the quaternions, and using the DCM from the numerical section, which used ode45 to propagate.

The last step of the homework was to plot the error history of the quaternion propagation methods, which I did using the given equations:

$$error = \delta\beta - [1,0,0,0]^T$$

where

$$\delta\beta = \beta_1 \otimes \beta_2^{-1}$$

where, β_1 is the analytically propagated quaternion, and β_2 is the (pitch-roll-yaw) from previous part converted to quaternions.

Integrating this into the code proved a challenge, but by using quatmultiply(), I was able to produce the delta beta equation and solve for error over the case of the time span given. The graph I produced for this is shown in Figure 8.

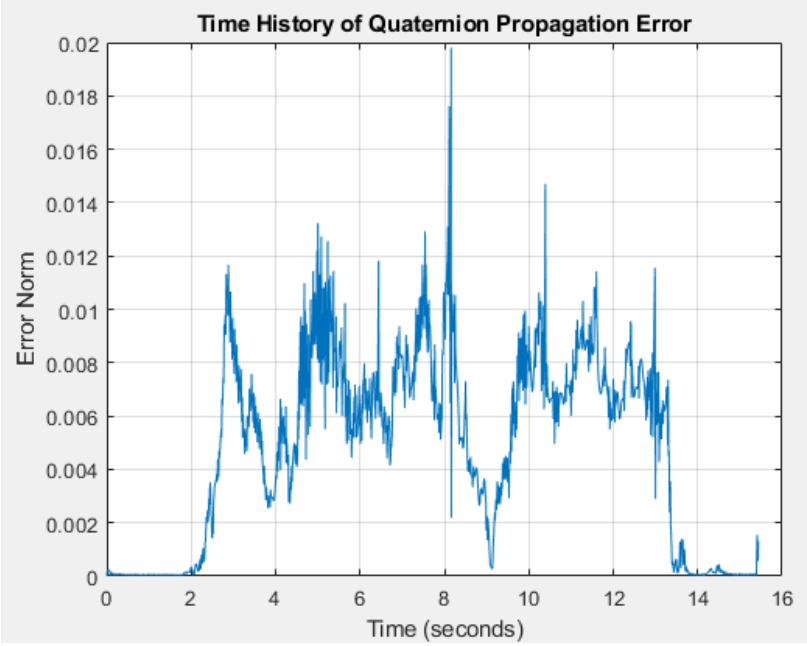

Figure 8. Time History of Quaternion Propagation Error graph

Upon analysis, the graph shows that the error is highest at about 8 seconds, which is around halfway through the time span. It shows a similar trend to Figure 5, which suggests the strong relationship between DCMs and quaternions, as they both approach zero at the beginning and end of the time span, while fluctuating in between at similar trends.

We also find that our error between numerical and analytical methods follow the same pattern, as that is what is being graphed in both cases. However, the propagation methods showcase strong trends together, as seen in Figures 6 and 7, where the Numerical and Analytical methods are trending almost identically. The error norm values are extremely low, as well and fit under 0.02 as the top bound for the graph.

The MATLAB code is copied and pasted, with proper formatting, on the following pages:

```
%% AERSP 450 - Homework #4
% Sunay Neelimathara - FA 24 - Professor Eapen

%% Problem 1
clc;
clear;
close all;

T = readtable('SensorData.csv');

wx = T.wx;
wy = T.wy;
wz = T.wz;

timeData = datetime(T.time, 'InputFormat', 'yyyy-MM-dd''T''HH:mm:ss.SSS''Z''',...
'TimeZone', 'UTC');

timeDifferences =  timeData - timeData(1);

time_diff = seconds(diff(timeDifferences));
average_interval = mean(time_diff);
frequency = 1 / average_interval;
fprintf('Sensor Frequency: %.2f Hz\n\n', frequency);

t = seconds(timeDifferences);

yaw0 = 30;
pitch0 = 70;
roll0 = 20;
theta1 = yaw0;
theta2 = pitch0;
theta3 = roll0;

% Find Euler angle sequence
% Yaw-Pitch-Roll is 3-2-1 Euler Angle Sequence


% Yaw-Pitch-Roll into DCM
% From our Euler Angle Relations:
C_ypr = [cosd(theta1)*cosd(theta2)  cosd(theta2)*sind(theta1)  -sind(theta2)
         sind(theta3)*sind(theta2)*cosd(theta1)-cosd(theta3)*sind(theta1)
sind(theta3)*sind(theta2)*sind(theta1)+cosd(theta3)*cosd(theta1)
sind(theta3)*cosd(theta2)
         cosd(theta3)*sind(theta2)*cosd(theta1)+sind(theta3)*sind(theta1)
cosd(theta3)*sind(theta2)*sind(theta1)-sind(theta3)*cosd(theta1)
cosd(theta3)*cosd(theta2)];

% Printing the calculated DCM
fprintf('DISPLAYING DCM: \n');
fprintf('\n The DCM for the yaw-pitch-roll angles (3-2-1 euler angle sequence) is as
follows: \n \n');
disp(C_ypr);
fprintf('\n');

% Verifying that DCM is valid, by seeing if its determinant is equal to 1.
```

```
fprintf('DCM VERIFICATION: \n \n');
verification_DCM = det(C_ypr);
fprintf('||DCM|| = ');
disp(verification_DCM);
fprintf('DCM is valid. \n \n \n');

% Yaw-Pitch-Roll into quaternion
% Using the equation from the roadmap, we can convert to quaternions:
beta0 = (1/2) * sqrt(C_ypr(1, 1) + C_ypr(2, 2) + C_ypr(3, 3) + 1);
beta1 = (C_ypr(2, 3) - C_ypr(3, 2)) / (4*beta0);
beta2 = (C_ypr(3, 1) - C_ypr(1, 3)) / (4*beta0);
beta3 = (C_ypr(1, 2) - C_ypr(2, 1)) / (4*beta0);

quat1 = [beta0, beta1, beta2, beta3]';

% Displaying the quaternions:
fprintf('DISPLAYING QUATERNIONS: \n');
fprintf('\n The quaternions for the yaw-pitch-roll angles (3-2-1 euler angle
sequence) are as follows: \n \n \t Beta0 = ');
disp(beta0);
fprintf('\t Beta1 = ');
disp(beta1);
fprintf('\t Beta2 = ');
disp(beta2);
fprintf('\t Beta3 = ');
disp(beta3);
fprintf('\n');


% Plot angular velocities as a function of time
% We are given the omegas, and t.
figure;
hold on;
title('\omega vs. time');
plot(t, wx);
plot(t, wy);
plot(t, wz);
hold off;
xlabel('time (seconds)');
ylabel('\omega (m/s)');
legend('\omega_{x}', '\omega_{y}', '\omega_{z}');
grid on;

% Comment on the nature of the data: (a) what is the frequency of the sensor output.
(b) are the sensor outputs at equal time intervals?
total_time = t(1544, :);
time_interval = size(t);
frequency = 1/total_time;

fprintf('\n TIME DIFFERENCES: \n');
%disp(t);

%% Problem 2

% Part 1
```

```
% Cdot_BN = -[w_tilde]*C_BN

omega = [wx, wy, wz];

w_tilde = @(w) [0 -w(3) w(2);
                w(3) 0 -w(1);
                -w(2) w(1) 0];

DCM_prop = @(time, C_BN) reshape(-w_tilde(interp1(t, omega, time, 'linear')') *
reshape(C_BN, 3, 3), [], 1);

C0 = C_ypr(:);

[t_out, C_ypr_ode] = ode45(@(time, C_BN) DCM_prop(time, C_BN), t, C0);

C_ypr_matrices = zeros(3, 3, length(t_out));
DCM_Valid = zeros(length(t_out));

for i = 1:length(t_out)
    C_ypr_matrices(:, :, i) = reshape(C_ypr_ode(i, :), 3, 3);
end

C_ypr_matrix = C_ypr_matrices(:, :, end);
DCM_Valid = det(C_ypr_matrix);


% Part 2
% C_BN(tk+1) = expm(-[w_tilde]*deltatk) C_BN(tk)

C_ypr_matrices_3 = zeros(3, 3, length(t));
C_ypr_matrices_3(:, :, 1) = C_ypr;

for k = 1:length(t)-1

    dt = t(k+1) - t(k);
    omega_k = omega(k, :);
    w_skew = w_tilde(omega_k);
    exp_w = expm(-w_skew * dt);
    C_ypr_matrices_3(:, :, k+1) = exp_w * C_ypr_matrices_3(:, :, k);

end

disp('Final Numerical DCM:')
disp(C_ypr_matrix);


disp('Final Analytical DCM:');
disp(C_ypr_matrices_3(:, :, end));
C_ypr_matrix_3_f = C_ypr_matrices_3(:, :, end);

% Part 3

identity_3 = eye(3);
error_tk = zeros(3, 3, length(t));
for k = 1:length(t)
```

```
    C_ypr_num = C_ypr_matrices(:, :, k);
    C_ypr_analytic = C_ypr_matrices_3(:, :, k);
    error_tk(:, :, k) = C_ypr_matrices(:, :, k) * C_ypr_matrices_3(:, :, k)' -
identity_3;
end

error_norm = zeros(length(t), 1);
for k = 1:length(t)
    error_norm(k) = norm(error_tk(:, :, k));
end

figure;
plot(t, error_norm);
xlabel('Time (s)');
ylabel('Error Norm');
title('Time History of DCM Error');
grid on;

% Part 4
yaw = zeros(1, length(t));
pitch = zeros(1, length(t));
roll = zeros(1, length(t));

for k = 1:length(t)
    C_ypr = C_ypr_matrices(:, :, k);
    roll(k) = atan2d(C_ypr(2, 3), C_ypr(3, 3));
    pitch(k) = atan2d(-C_ypr(1, 3), sqrt(C_ypr(1, 1)^2 + C_ypr(1, 2)^2));
    yaw(k) = atan2d(C_ypr(1, 2), C_ypr(1, 1));
end

figure;
subplot(2, 1, 1);
hold on;
plot(t, yaw);
plot(t, pitch);
plot(t, roll);
xlabel('Time (s)');
ylabel('Yaw (rad)');
title('Yaw-Pitch-Roll vs Time - Numerical');
legend('yaw', 'pitch', 'roll')
grid on;
hold off;


for k = 1:length(t)
    C_ypr = C_ypr_matrices_3(:, :, k);
    roll(k) = atan2d(C_ypr(2, 3), C_ypr(3, 3));
    pitch(k) = atan2d(-C_ypr(1, 3), sqrt(C_ypr(1, 1)^2 + C_ypr(1, 2)^2));
    yaw(k) = atan2d(C_ypr(1, 2), C_ypr(1, 1));
end

subplot(2, 1, 2);
hold on;
plot(t, yaw);
plot(t, pitch);
```

```
plot(t, roll);
xlabel('Time (s)');
ylabel('Yaw (rad)');
title('Yaw-Pitch-Roll vs Time - Analytical');
legend('yaw', 'pitch', 'roll')
grid on;
hold off;

%% Problem 3
% Part 1

quat0 = [beta0, beta1, beta2, beta3];
beta_calc = zeros(length(t), length(quat0));
beta_calc(1, :) = quat0;

B_omega = @(w) [0 -w(1) -w(2) -w(3);
              w(1) 0 w(3) -w(2);
              w(2) -w(3) 0 w(1);
              w(3) w(2) -w(1) 0];

omega_transpose = omega';

for k = 1:length(t)-1

    dt = t(k+1) - t(k);

    w = omega_transpose(:, k);
    B_w = B_omega(w);
    phi_quat = expm(0.5 * B_w * dt);
    beta_calc(k+1, :) = (phi_quat * beta_calc(k, :)')';

end

% Part 2
quat_const = ones(length(t));
figure;
subplot(2, 1, 1);
hold on;
plot(t, beta_calc(:, 1));
plot(t, beta_calc(:, 2));
plot(t, beta_calc(:, 3));
plot(t, beta_calc(:, 4));
plot(t, quat_const, 'g');
legend('\beta_{0}', '\beta_{1}', '\beta_{2}', '\beta_{3}', 'Quaternion Constant:
\beta = 1');
xlabel('time (seconds)');
title('Quaternion Propagation from Equations of Motion');
hold off;
grid on;

% Part 3
% yaw-pitch-roll from previous section analytical to quaternions
beta0_f = zeros(1, length(t));
beta1_f = zeros(1, length(t));
beta2_f = zeros(1, length(t));
```

```
beta3_f = zeros(1, length(t));
% test = zeros(4, length(t));

quaternion_array = zeros(4, length(t));
quaternion_array(:, 1) = quat1;
norm_quat = zeros(1, length(t));
norm_quat(1) = norm(quaternion_array);

for k= 1:length(t)-1
    C_ypr = C_ypr_matrices_3(:, :, k);
    beta0_f = (1/2) * sqrt(C_ypr(1, 1) + C_ypr(2, 2) + C_ypr(3, 3) + 1);
    beta1_f = (C_ypr(2, 3) - C_ypr(3, 2)) / (4*beta0_f);
    beta2_f = (C_ypr(3, 1) - C_ypr(1, 3)) / (4*beta0_f);
    beta3_f = (C_ypr(1, 2) - C_ypr(2, 1)) / (4*beta0_f);
    quaternion_array(:, k+1) = [beta0_f; beta1_f; beta2_f; beta3_f];
    if dot(quaternion_array(:, k), quaternion_array(:, k+1)) < 0
        quaternion_array(:, k+1) = -quaternion_array(:, k+1);
    end
    quaternion_array(:, k+1) = quaternion_array(:, k+1) / norm(quaternion_array(:,
k+1));
    norm_quat(k+1) = norm(quaternion_array(:, k+1));
end

subplot(2, 1, 2);
hold on;
plot(t, quaternion_array(1, :));
plot(t, quaternion_array(2, :));
plot(t, quaternion_array(3, :));
plot(t, quaternion_array(4, :));
plot(t, norm_quat, 'g');
legend('\beta_{0}', '\beta_{1}', '\beta_{2}', '\beta_{3}', 'Quaternion Constant:
\beta = 1');
xlabel('time (seconds)');
title('Quaternions from Analytical Propagation of Yaw-Pitch-Roll Angles');
hold off;
grid on;

% Part 4
% Error
error_quaternion = zeros(4, length(t));
error_norm_quat = zeros(1, length(t));

for k = 1:length(t)
    % Analytical quaternion
    beta1f = beta_calc(k, :); % Row vector
    beta2f = quaternion_array(:, k)'; % Convert column to row vector
    % Quaternion inverse (conjugate)
    beta2_inv = [beta2f(1), -beta2f(2:4)];
    % Quaternion multiplication
    delta_beta = quatmultiply(beta1f, beta2_inv); % Output is a row vector
    delta_beta = delta_beta'; % Convert to column vector
    % Quaternion error
    error_quaternion(:, k) = delta_beta - [1; 0; 0; 0]; % Ensure column alignment
end
```

```
% Compute norm of the quaternion error
for k = 1:length(t)
    error_norm_quat(k) = norm(error_quaternion(:, k));
end

% Plot the error norm
figure;
plot(t, error_norm_quat);
xlabel('Time (seconds)');
ylabel('Error Norm');
title('Time History of Quaternion Propagation Error');
grid on;
```