**Problem I**

1. Given Yaw = 30°, Pitch = 70°, and Roll = 20°

This is a 3-2-1 rotation because the standard notation is that roll is a rotation about , pitch is a rotation about , and yaw is a rotation about . Also, it is known that “the standard yaw-pitch-roll (ψ; θ; φ) angles are the (3-2-1) set of Euler angles” (Junkins and Schaub, p. 87).

1. The DCM for a 3-2-1 rotation is known and shown in Figure 1.

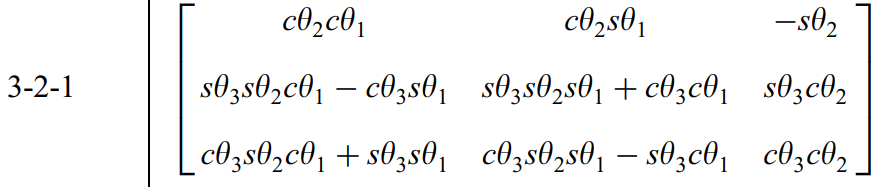


Figure 1: 3-2-1 rotation matrix. Source: Junkins and Schaub, Appendix B

where is the yaw, is the pitch, and is the roll. Plugging the numbers in yields the following DCM:

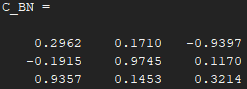


Figure 2: DCM that describes the attitude of the body WRT the inertial frame

1. A math equations on a white background

   Description automatically generatedConverting this DCM into a quaternion using Sheppard’s algorithm:

Figure 3: Conversion equations from DCM to Quaternion using Sheppard’s method.  
Source: Junkins and Schaub, p. 105

Plugging in the DCM from part 2 yields

1. The angular velocity was given in the *SensorData.csv* file, and the data was extracted using the provided pseudocode. This data is plotted blow:

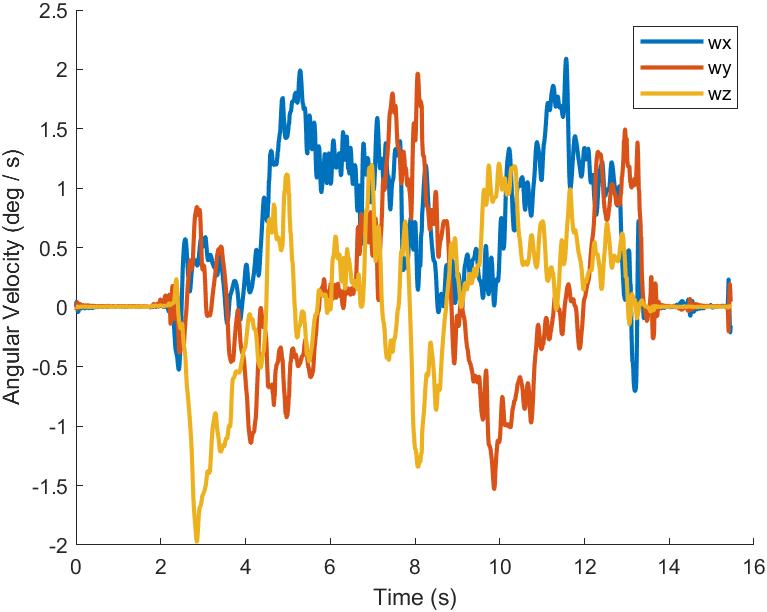


Figure 4: Angular velocity components plotted over time

1. The frequency of the sensor input can be found by looking at the values of the time variable, *t*, in Matlab. It shows that data is, generally, being taken every 0.01 seconds. However, it is not always at equal time intervals; Sometimes the timing is off by 0.001 seconds. Below is a snippet of some of the time values of *t* that shows this:

|  |  |
| --- | --- |
| Index | Value |
| 162 | 1.6100 |
| 163 | 1.6200 |
| 164 | 1.6310 |
| 165 | 1.6410 |
| 166 | 1.6500 |
| 167 | 1.6600 |

**Problem 1 Code**

% This is the code for Aersp 450, HW 4, Question I

% Made by Nicholas Luis (PSU ID 930841391)

clc

clear

%% Provided Skeleton Code

T = readtable('SensorData.csv');

wx = T.wx;

wy = T.wy;

wz = T.wz;

% Step 1: Convert the time strings into datetime format

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

'TimeZone', 'UTC');

% Step 2: Calculate time differences from the first time in the list

timeDifferences = timeData - timeData(1);

% Step 3: Convert the differences to seconds

t = seconds(timeDifferences);

%%

% Given euler angle rotations

theta1 = 30; % yaw

theta2 = 70; % pitch

theta3 = 20; % theta3l

% DCM rotation based on a 3-2-1 rotation

C\_BN = [cosd(theta2)\*cosd(theta1), 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);

]

DCMcheck(C\_BN);

% Quaternion based on the DCM matrix using Sheppard Algo

Beta = SheppardAlgo(C\_BN)

% Plotting the angular velocities as a function of time

figure(1)

hold on

plot(t, wx, LineWidth=2)

plot(t, wy, LineWidth=2)

plot(t, wz, LineWidth=2)

xlabel("Time (s)")

ylabel("Angular Velocity (deg / s)")

legend('wx', 'wy', 'wz')

hold off

exportgraphics(gca,"HW4\_Problem1\_AngVeloPlots.jpg");

%% Functions

function isDCM = DCMcheck(A)

% This function checks if a matrix is a DCM

isDCM = true;

% Checks if the rows and columns are unit vectors

for i = 1:3

if (round(norm(A(i,:)), 10) ~= 1)

isDCM = false;

fprintf("The DCM is not valid!");

return;

end

if (round(norm(A(:,i)), 10) ~= 1)

isDCM = false;

fprintf("The DCM is not valid!");

return;

end

end

% Checks if the DCM is orthonormal

if (round(A\*A',10) ~= eye(3))

isDCM = false;

fprintf("The DCM is not valid!");

return;

end

end

function BetaVec = SheppardAlgo(C)

% This funciton inputs some matrix C and does Sheppard's algorithm to

% compute the quaternion

% Equation 3.95

B0 = sqrt(0.25\*(1+trace(C)));

B1 = sqrt(0.25\*(1+2\*C(1,1)-trace(C)));

B2 = sqrt(0.25\*(1+2\*C(2,2)-trace(C)));

B3 = sqrt(0.25\*(1+2\*C(3,3)-trace(C)));

BetaVec = [B0; B1; B2; B3];

% This part of Sheppard's algorithm leads to sum(B\_i ^2) < 1 ???

%{

biggestB = max(BetaVec);

% Equation 3.96

if (B0 == biggestB)

BetaVec(1) = 0.25\*(C(2,3)-C(3,2))/B0;

BetaVec(2) = 0.25\*(C(3,1)-C(1,3))/B0;

BetaVec(3) = 0.25\*(C(1,2)-C(2,1))/B0;

return;

elseif (B1 == biggestB)

BetaVec(0) = 0.25\*(C(2,3)-C(3,2))/B1;

BetaVec(2) = 0.25\*(C(1,2)+C(2,1))/B1;

BetaVec(3) = 0.25\*(C(3,1)+C(1,3))/B1;

return;

elseif (B2 == biggestB)

BetaVec(0) = 0.25\*(C(3,1)-C(1,3))/B2;

BetaVec(1) = 0.25\*(C(1,2)+C(2,1))/B2;

BetaVec(3) = 0.25\*(C(2,3)+C(3,2))/B2;

return;

elseif (B3 == biggestB)

BetaVec(0) = 0.25\*(C(1,2)-C(2,1))/B3;

BetaVec(1) = 0.25\*(C(3,1)+C(1,3))/B3;

BetaVec(2) = 0.25\*(C(2,3)+C(3,2))/B3;

return;

end

%}

end

. .

**Problem II**

1. Numerically propagating the DCM

This was done using the following equation:

At every time step, the rate of change in the DCM, , was calculated and multiplied by the timestep to get the absolute change of the DCM. Then, this change was added to the DCM, , to get the new DCM at that timestep. This process repeated for every timestep.

1. Analytically propagating the DCM

This was done using the following equation:

At every time step, the new DCM was calculated using that equation, and it uses the DCM of the previous timestep.

1. Plotting the time history of the error of the DCM

The error DCM was calculated at every timestep using:



where and were obtained from parts 1 and 2, respectively. Then, the *norm( )* function was used to obtain the magnitude of the error, which is plotted below:

A graph showing a blue line

Description automatically generated

1. Plotting the Yaw-Pitch-Roll vs Time

The DCM contains the all the angles. It can be extracted using these equations:

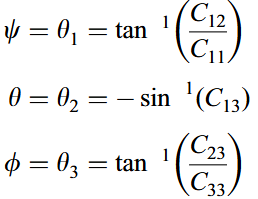
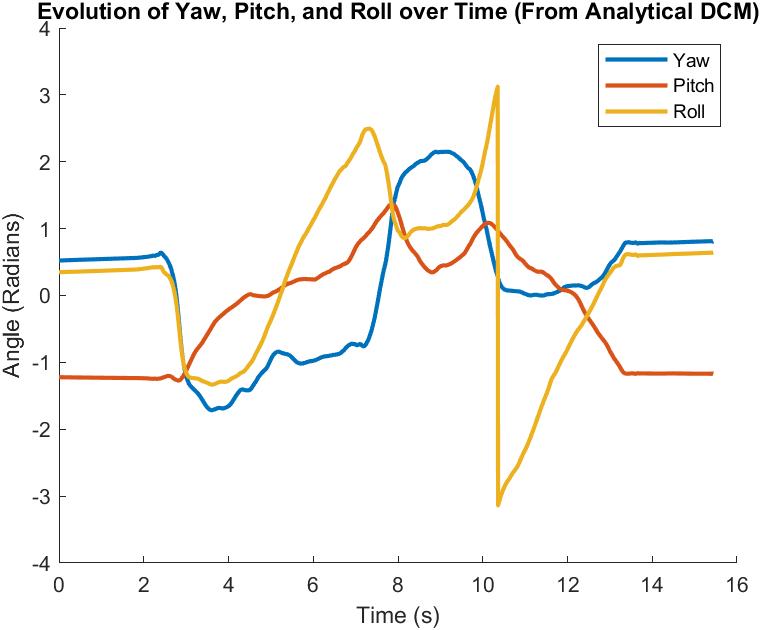


Figure 5 Conversion equations from DCM to Euler Angle  
 Source: Junkins and Schaub, p. 87

These angles were calculated at every time step because the DCM is changing. It was also calculated using the two different DCM’s, one from the numerical solution and the other from the analytic solution. These plots are shown side-by-side here. There are nearly identical; Any differences are negligible because the error is very small, as seen in question 3’s error history plot.

A graph of different colored lines

Description automatically generated

**Problem 2 Code**

% This is the code for Aersp 450, HW 4, Question II

% Made by Nicholas Luis (PSU ID 930841391)

clc

clear

%% Import Data

T = readtable('SensorData.csv');

wx = T.wx; % Roll rate

wy = T.wy; % Pitch rate

wz = T.wz; % Yaw rate

% Step 1: Convert the time strings into datetime format

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

'TimeZone', 'UTC');

% Step 2: Calculate time differences from the first time in the list

timeDifferences = timeData - timeData(1);

% Step 3: Convert the differences to seconds

t = seconds(timeDifferences);

% Given euler angle rotations

theta1 = 30; % yaw

theta2 = 70; % pitch

theta3 = 20; % roll

% DCM rotation based on a 3-2-1 rotation

C\_BN\_original = [cosd(theta2)\*cosd(theta1), 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);

];

%% Numerically Propogating the DCM at every timestep

C\_BN = C\_BN\_original; % Creates a copy of the original DCM

C\_hist\_numerical = zeros(length(t)-1,9); % Stores the DCM at each timestep

C\_BN\_dot = zeros(3,3);

for i=1:length(t)-1

% Saving the DCM as a single 9-element line

C\_hist\_numerical(i,:) = mat2row(C\_BN);

% Propogating DCM

omegaTilda = skewSymmetric([wx(i), wy(i), wz(i)]); % Creates the skew symmetric matrix of the angular velocity at a given time

C\_BN\_dot = - omegaTilda \* C\_BN;

C\_BN = C\_BN + (C\_BN\_dot\*(t(i+1)-t(i)));

end

%% Analytically Propogating the DCM at every timestep

C\_BN = C\_BN\_original; % Creates a copy of the original DCM

C\_hist\_analytic = zeros(length(t)-1,9); % Stores the DCM at each timestep

for j=1:length(t)-1

C\_hist\_analytic(j,:) = mat2row(C\_BN);

omegaTilda = skewSymmetric([wx(j), wy(j), wz(j)]); % Creates the skew symmetric matrix of the angular velocity at a given time

% Propagating DCM

C\_BN = expm( -omegaTilda.\*(t(j+1)-t(j)) ) \* C\_BN;

end

%% Analytical vs Numerical Comparisons

error\_hist = zeros(length(C\_hist\_numerical)-1,1); % Saves the error at every timestep

for i = 1:length(C\_hist\_numerical)

% Gets the DCM at the given time step

C\_numeric = row2mat(C\_hist\_numerical(i,:));

C\_analyic = row2mat(C\_hist\_analytic(i,:));

% Calculates error between two DCM's

error\_hist(i) = norm(C\_numeric\*C\_analyic'-eye(3));

end

% Plotting

figure(1)

hold on

plot(t(1:length(t)-1), error\_hist, LineWidth=2)

title('Time History of the Error' )

xlabel("Time (s)")

ylabel("Error magnitude")

hold off

exportgraphics(gca,"HW4\_Problem2\_ErrorPlot.jpg");

%% Yaw-Pitch-Roll vs Time

% Getting Euler Angle values from the DCMs

numericAngleHistory = zeros(length(t)-1, 3);

analyticAngleHistory = zeros(length(t)-1, 3);

for i = 1:length(t)-1

% Numerical Data

C\_numeric = row2mat(C\_hist\_numerical(i,:));

theta1 = atan2(C\_numeric(2,1), C\_numeric(1,1)); % Yaw

theta2 = asin(C\_numeric(3,1)); % Pitch

theta3 = atan2(C\_numeric(3,2), C\_numeric(3,3)); % Roll

numericAngleHistory(i,:) = [theta1, theta2, theta3]; % Saves yaw, pitch, and roll at this time step

% Analytical Data

C\_analytic = row2mat(C\_hist\_analytic(i,:));

theta1 = atan2(C\_analytic(2,1), C\_analytic(1,1)); % Yaw

theta2 = asin(C\_analytic(3,1)); % Pitch

theta3 = atan2(C\_analytic(3,2), C\_analytic(3,3)); % Roll

analyticAngleHistory(i,:) = [theta1, theta2, theta3]; % Saves yaw, pitch, and roll at this time step

end

% Plotting values

figure(2)

hold on

plot(t(1:length(t)-1), numericAngleHistory(:,1), LineWidth=2)

plot(t(1:length(t)-1), numericAngleHistory(:,2), LineWidth=2)

plot(t(1:length(t)-1), numericAngleHistory(:,3), LineWidth=2)

legend('Yaw ', 'Pitch', 'Roll')

title('Evolution of Yaw, Pitch, and Roll over Time (From Numerical DCM)' )

xlabel("Time (s)")

ylabel("Angle (Radians)")

hold off

exportgraphics(gca,"HW4\_Problem2\_EulerAnglesPlot.jpg");

figure(3)

hold on

plot(t(1:length(t)-1), analyticAngleHistory(:,1), LineWidth=2)

plot(t(1:length(t)-1), analyticAngleHistory(:,2), LineWidth=2)

plot(t(1:length(t)-1), analyticAngleHistory(:,3), LineWidth=2)

legend('Yaw ', 'Pitch', 'Roll')

title('Evolution of Yaw, Pitch, and Roll over Time (From Analytical DCM)' )

xlabel("Time (s)")

ylabel("Angle (Radians)")

hold off

exportgraphics(gca,"HW4\_Problem2\_EulerAnglesPlot2.jpg");

%% Functions

function matrixTilda = skewSymmetric(vec)

% This function inputs a vector and returns a skew symmetrix matrix

matrixTilda = [0, -vec(3), vec(2);...

vec(3), 0, -vec(1);

-vec(2), vec(1), 0;];

end

function rowMatrix = mat2row(inputMat)

% This function converts a 3x3 to a 9x1

rowMatrix = zeros(1,9);

ctr = 1;

for iter = 1:3

for jter = 1:3

rowMatrix(ctr) = inputMat(iter,jter);

ctr = ctr + 1;

end

end

end

function matrixOutput = row2mat(inputVec)

% This funciton inputs a 9x1 vector and converts it to a 3x3 matrix

matrixOutput = zeros(3,3);

ctr = 1;

for jter = 1:3

for iter = 1:3

matrixOutput(iter,jter) = inputVec(ctr);

ctr = ctr + 1;

end

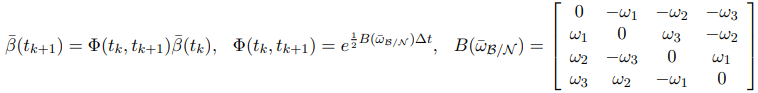
end

end

. .

**Problem III**

1. Analytically propagate the Quaternions

Analytic propagation was done using this equation:

This was done at every timestep.

1. Plotting the quaternions and showing that they are satisfied at every timestep.

Once again, to plot a vector over time the function *norm( )* was used to get the magnitude of the quaternion at each time step. The quaternion is satisfied when the sum of the Beta components squared is equal to one:

This was calculated at each iteration and is plotted below:

A line graph with numbers and numbers

Description automatically generatedAs you can see, it is always at 1, which is good. It means that the constraint holds true at every timestep, and everything was done correctly.

1. Converting the Analytically-obtained Roll, Pitch, and Yaw into Quaternions.

A diagram of mathematical equations

Description automatically generatedConsulting the attitude description roadmap, we first convert these Euler Angles into a DCM, which is the same process as in part I.1. We then convert the DCM into a quaternion, which is the same process that was used in part I.3.

Figure 6- Roadmap that shows how to convert from one attitude description to another. Source: Dr. Eapen, reference files

The angles that were being converted is the analytically-obtained Euler Angles from part II.2. Each of these sets of Euler Angles were converted to DCM at every time step. Then every DCM was converted to Quaternions at every time step.

1. Comparing the quaternions obtained via analytical method to the quaternion obtained by converting the roll, pitch, and yaw values.

This was done using the following equation:

, where .

Error is, once again, multi-dimensional so the *norm( )* was taken in order to generate a 2D plot w.r.t time. Below is the time history of this error:

A graph showing a line of a graph

Description automatically generated with medium confidence

**Problem 3 Code**

% This is the code for Aersp 450, HW 4, Question II

% Made by Nicholas Luis (PSU ID 930841391)

clc

clear

close all

%% Import Data

T = readtable('SensorData.csv');

wx = T.wx; % Roll rate (omega 3)

wy = T.wy; % Pitch rate (omega 2)

wz = T.wz; % Yaw rate (omega 1)

% Step 1: Convert the time strings into datetime format

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

'TimeZone', 'UTC');

% Step 2: Calculate time differences from the first time in the list

timeDifferences = timeData - timeData(1);

% Step 3: Convert the differences to seconds

t = seconds(timeDifferences);

% Given euler angle rotations

theta1 = 30; % yaw

theta2 = 70; % pitch

theta3 = 20; % roll

% DCM rotation based on a 3-2-1 rotation

C\_BN\_original = [cosd(theta2)\*cosd(theta1), 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);

];

% Initial Beta obtained (Copied from Part I)

Beta = [0.8050, -0.0088, 0.5824, 0.1126];

% Analytic Propagation (Copied from Part II)

C\_BN = C\_BN\_original; % Creates a copy of the original DCM

C\_hist\_analytic = zeros(length(t)-1,9); % Stores the DCM at each timestep

for j=1:length(t)-1

C\_hist\_analytic(j,:) = mat2row(C\_BN);

omegaTilda = skewSymmetric3x3([wx(j), wy(j), wz(j)]); % Creates the skew symmetric matrix of the angular velocity at a given time

% Propagating DCM

C\_BN = expm( -omegaTilda.\*(t(j+1)-t(j)) ) \* C\_BN;

end

% Getting Yaw, Pitch, and Roll values from the DCM (from Part II)

analyticAngleHistory = zeros(length(t)-1, 3);

for i = 1:length(t)-1

% Analytical Data

C\_analytic = row2mat(C\_hist\_analytic(i,:));

theta1 = atan2(C\_analytic(2,1), C\_analytic(1,1)); % Yaw

theta2 = asin(C\_analytic(3,1)); % Pitch

theta3 = atan2(C\_analytic(3,2), C\_analytic(3,3)); % Roll

analyticAngleHistory(i,:) = [theta1, theta2, theta3]; % Saves yaw, pitch, and roll at this time step

end

%% Propagating the quaternion

B\_hist = zeros(length(t)-1, 5); % Saves the quaternion vectors at every timestep (the last index is the magnitude)

B\_hist(1,:) = [Beta, norm(Beta)];

for i = 1:length(t)-1

B = skewSymmetric([wx(i),wy(i),wz(i)]);

Phi = expm(0.5\*B\*(t(i+1)-t(i)));

Bnew = Phi\*B\_hist(i,1:4)'; % Propagation

B\_hist(i+1, :) = [Bnew', norm(Bnew)];

end

% Checking if quaternion constraint is satisfied

constraintCheck = zeros(length(t), 1);

for j = 1:length(t)

constraintCheck(j) = (B\_hist(j,1)^2 + B\_hist(j,2)^2 + B\_hist(j,3)^2 + B\_hist(j,4)^2);

end

% Plotting

figure(1)

hold on

plot(t(1:length(t)), round(constraintCheck,3), LineWidth=2)

title('Constraint check of Beta vectors' )

xlabel("Time (s)")

ylabel("Sum of (B\_i^2)")

hold off

exportgraphics(gca,"HW4\_Problem3\_ConstraintCheck.jpg");

%% Converting Analytic Roll-Pitch-Yaw to Quaternion

% Converting Euler Angles to DCM at every timestep

DCM\_hist = zeros(length(t)-1,9); % Creates a Nx9 matrix; Each 9-element row stores all the values of the DCM

for k = 1:length(t)-1

DCM\_hist(k,:) = mat2row( EulerAngles2DCM(analyticAngleHistory(k,:)) );

end

% Converting DCM to Quaternion at every timestep

B\_Euler\_hist = zeros(length(t)-1,4); % Each row in the matrix stores each of the 4 values of the quaternion

for L = 1:length(t)-1

tempDCM = row2mat(DCM\_hist(L,:)); % Convert the DCM row back into a 3x3

B\_Euler\_hist(L,:) = DCM2Quaternion(tempDCM); % Converts the DCM into a Quaternion

end

%% Comparing the Analytic Quaternions to the Euler-Angle-Converted Quaternions

error = zeros(length(t)-1,1); % Vector to record the errors at each time step

for m = 1:length(t)-1

deltaBeta = quaternionMultiplication(B\_hist(m,1:4), B\_Euler\_hist(m,:));

error(m) = norm(deltaBeta - [1,0,0,0]');

end

figure(2)

hold on

plot(t(1:length(t)-1), error, LineWidth=2)

title('Error History of the Quaternions obtained from Analyitcal vs Euler-Angle Methods')

xlabel("Time (s)")

ylabel("Magnitude of Error")

hold off

exportgraphics(gca,"HW4\_Problem3\_QuaternionErrorComparison.jpg");

%% Functions

function matrixTilda = skewSymmetric(vec)

% This function inputs a vector and returns a skew symmetrix matrix

matrixTilda = [0, -vec(1), -vec(2), -vec(3);...

vec(1), 0, vec(3), -vec(2); ...

vec(2), -vec(3), 0, vec(1); ...

vec(3), vec(2), -vec(1), 0;];

end

function matrixTilda = skewSymmetric3x3(vec)

% This function inputs a vector and returns a skew symmetrix matrix

matrixTilda = [0, -vec(3), vec(2);...

vec(3), 0, -vec(1);

-vec(2), vec(1), 0;];

end

function DCM = EulerAngles2DCM(EulerAngles)

% This function converts Euler Angles to a DCM (from Part I.1)

theta1 = EulerAngles(1);

theta2 = EulerAngles(2);

theta3 = EulerAngles(3);

DCM = [cosd(theta2)\*cosd(theta1), 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);

];

end

function BetaVec = DCM2Quaternion(C)

% This funciton inputs some 3x3 matrix C and computes the quaternion (from Part I.3)

% Equation 3.95

B0 = sqrt(0.25\*(1+trace(C)));

B1 = sqrt(0.25\*(1+2\*C(1,1)-trace(C)));

B2 = sqrt(0.25\*(1+2\*C(2,2)-trace(C)));

B3 = sqrt(0.25\*(1+2\*C(3,3)-trace(C)));

BetaVec = [B0; B1; B2; B3];

end

function rowMatrix = mat2row(inputMat)

% This function converts a 3x3 to a 9x1

rowMatrix = zeros(1,9);

ctr = 1;

for iter = 1:3

for jter = 1:3

rowMatrix(ctr) = inputMat(iter,jter);

ctr = ctr + 1;

end

end

end

function matrixOutput = row2mat(inputVec)

% This funciton inputs a 9x1 vector and converts it to a 3x3 matrix

matrixOutput = zeros(3,3);

ctr = 1;

for jter = 1:3

for iter = 1:3

matrixOutput(iter,jter) = inputVec(ctr);

ctr = ctr + 1;

end

end

end

function beta = quaternionMultiplication(B1, B2)

% This function performs quaternion multiplicaiton

B1\_tilda = skewSymmetric(B1);

beta = B1\_tilda\*B2';

end

References:

Junkins, J., Schaub, H., *Analytical Mechanics of Space Systems, Fourth Edition (AIAA Education Series)*, AIAA American Institute of Aeronautics & Ast, April 2018