Homework 2 Problem 8

AE403 - Spring 2018

Emilio R. Gordon

The plots required for part A and part B are shown in the following pages. The initial conditions for this problem were defined to be

- **Initial Rotation Matrix**

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

- **Initial Euler Angles**

$$\begin{bmatrix} 0 \\ \frac{\pi}{2} \\ 0 \end{bmatrix}$$

- **Initial Quaternion**

$$Q = (q_o, q) = \left(\frac{1}{\sqrt{2}}, \begin{bmatrix} 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix} \right)$$

A function begins by defining a time interval which we are saying is 6 seconds. After which, we define the Initial conditions as mentioned above. After which ODE45 is utilized to generate the state values of Rotation matrix, Euler Angles and Quaternions.

```
% DEFINE THE TIME INTERVAL
tMax = 6;
tRate = 30;
t = linspace(0,tMax,tMax*tRate);

%DEFINE THE INITIAL CONDITIONS
xH0 = [0;pi/2;0];
R0 = [0 0 1; 0 1 0; -1 0 0];
xQ0 = [1/sqrt(2);0;1/sqrt(2);0];

% INTEGRATE THE ANGULAR RATE EQUATIONS
[t,xR] = ode45(@fR,t,RtoX(R0));
[t,xH] = ode45(@fH,t,xH0);
[t,xQ] = ode45(@fQ,t,xQ0);
```

Figure 1: ODE45 and Setup

After ODE45 has completed, the final states can be gathered by the following script.

```
%Collect State at t=6 seconds
finalR = xR(end,:)
finalH = xH(end,:)
finalQ = xQ(end,:)
```

Figure 2: State at t=6 seconds

From this, it appears that:

- **Rotation Matrix at 6 Seconds**

$$\begin{bmatrix} -0.9135 & -0.1261 & 0.3869 \\ -0.2036 & -0.6817 & -0.7027 \\ 0.3524 & -0.7206 & 0.5971 \end{bmatrix}$$

- **Euler Angles at 6 Seconds**

$$\begin{bmatrix} 7.1425 \\ 0.3985 \\ -3.2714 \end{bmatrix}$$

- **Quaternion at 6 Seconds**

$$Q = (q_o, q) = \left(-0.0216, \begin{bmatrix} 0.2069 \\ -0.3985 \\ 0.8934 \end{bmatrix} \right)$$

Which only partially answers part c.

Part C further ask to convert our final Euler Angles and Quaternion into a rotation matrix. This can easily be done in Matlab as shown in Figure 3.

```
% CODE TO CONVERT FROM XYZ EULER ANGLES TO A ROTATION MATRIX
function R=HtoR(H)
        theta1 = H(1);
        theta2 = H(2);
        theta3 = H(3);

        R11 = cos(theta2)*cos(theta3);
        R12 = -cos(theta2)*sin(theta3);
        R13 = sin(theta2);

        R21 = sin(theta1)*sin(theta2)*cos(theta3)+cos(theta1)*sin(theta3);
        R22 = -sin(theta1)*sin(theta2)*sin(theta3)+cos(theta1)*cos(theta3);
        R23 = -sin(theta1)*cos(theta2);

        R31 = -cos(theta1)*sin(theta2)*cos(theta3)+sin(theta1)*sin(theta3);
        R32 = cos(theta1)*sin(theta2)*sin(theta3)+sin(theta1)*cos(theta3);
        R33 = cos(theta1)*cos(theta2);

R=[R11 R12 R13; R21 R22 R23; R31 R32 R33];

% CODE TO CONVERT FROM A QUATERNION TO A ROTATION MATRIX
function R=QtoR(q0,q)
        qhat = [0 -q(3) q(2); q(3) 0 -q(1); -q(2) q(1) 0];

R = (q0^2 - q'*q)*eye(3)+2*q*q'+2*q0*qhat;
```

Figure 3: Functions for Conversions

This provides the solution:

- **Rotation Matrix at 6 Seconds**

$$\begin{bmatrix} -0.9135 & -0.1261 & 0.3869 \\ -0.2036 & -0.6817 & -0.7027 \\ 0.3524 & -0.7206 & 0.5971 \end{bmatrix}$$

- **Rotation Matrix from Euler Angles at 6 Seconds**

$$\begin{bmatrix} -0.9139 & -0.1193 & 0.3881 \\ -0.2069 & -0.6855 & -0.6980 \\ 0.3493 & -0.7182 & 0.6018 \end{bmatrix}$$

- **Rotation Matrix from Quaternion at 6 Seconds**

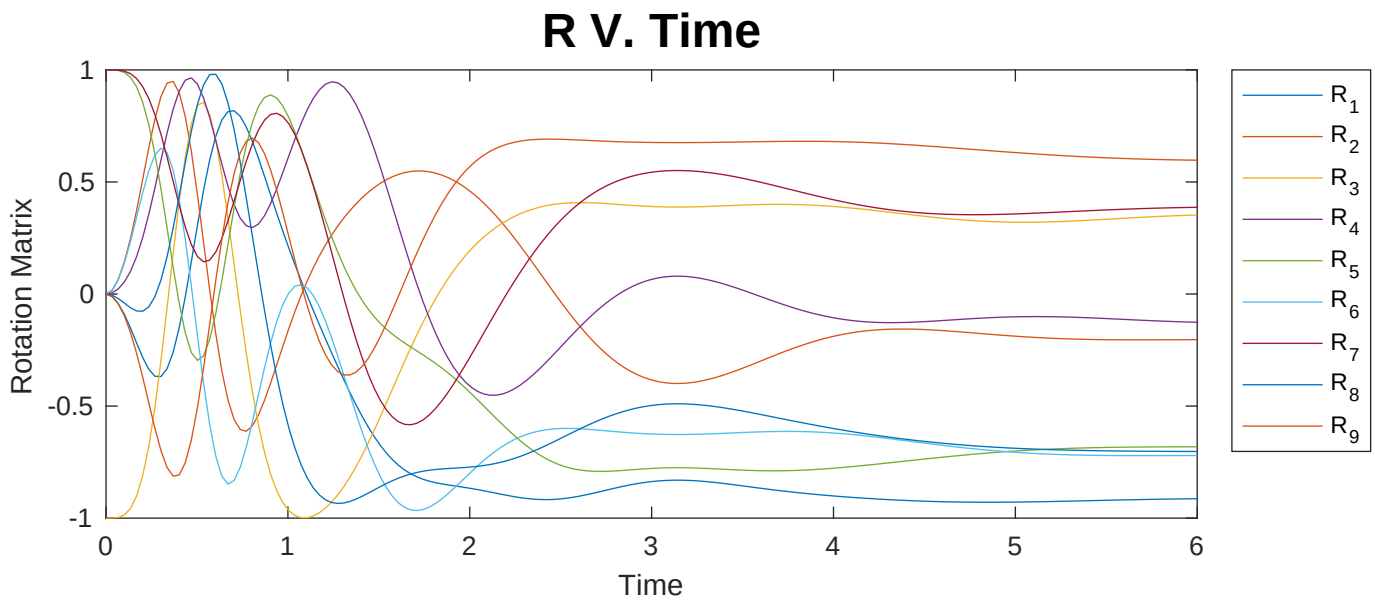
$$\begin{bmatrix} -0.9138 & -0.1263 & 0.3870 \\ -0.2036 & -0.6818 & -0.7031 \\ 0.3525 & -0.7211 & 0.5971 \end{bmatrix}$$

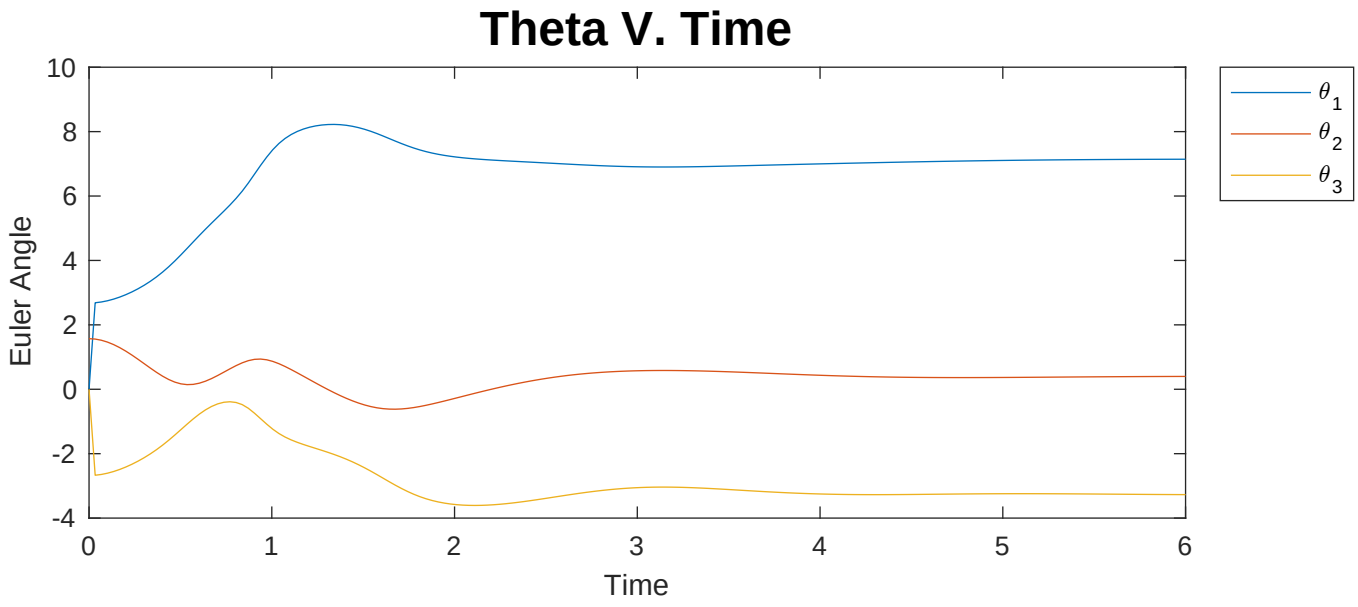
The rotation matrices derived from each coordinate system resulted in relatively the same matrix. That said, there is some error due to numerical rounding. This confirms our earlier analysis and work since, regardless of approach, we arrived at the same matrix.

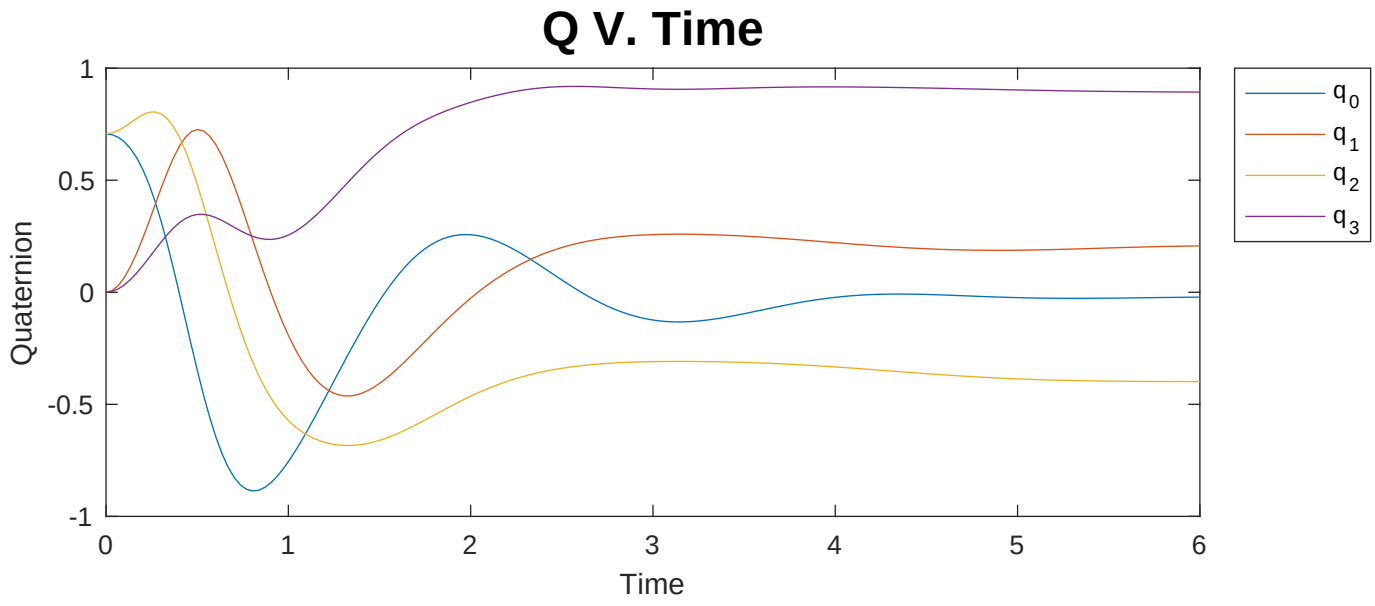
Every approach has its pros and cons as follows:

Name	Pros	Cons
Rotation Matrix	Always provides information on the state No singularity	Computational heavy 9 different elements to keep track of Not intuitive (opinion)
Euler Angles	Only 3 elements to work with. Computationally efficient Intuitive (opinion)	Singularity or "Gimbal Lock"
Quaternion	Computationally inexpensive Only 4 elements to work with No singularity Intuitive (opinion)	N/A

Plotting the states over time, the plots on the following page are produced. Notice how the states tend to reach a slope of zero towards the end. This tells us that the spacecraft is reaching a stable orientation. For complete code, please check compass.


R V. Time
Rotation Matrix
Time
R1
R2
R3
R4
R5
R6
R7
R8
R9


Theta V. Time
Euler Angle
Time
θ1
θ2
θ3

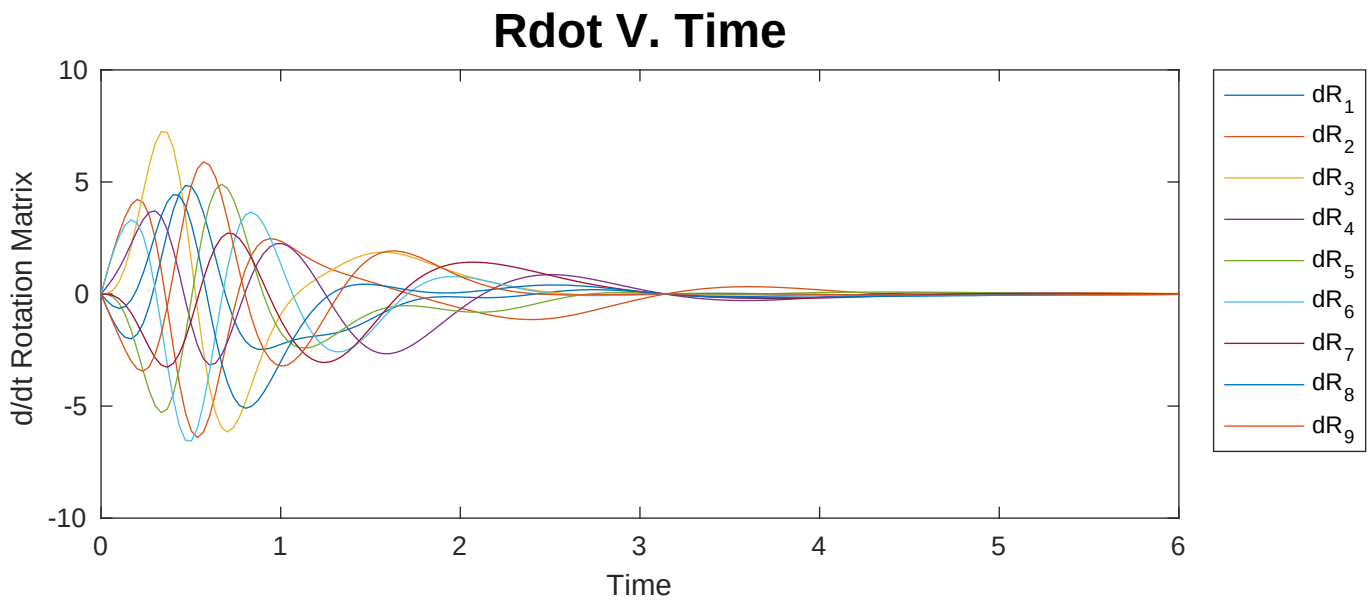

Q V. Time
Quaternion
Time
q0
q1
q2
q3

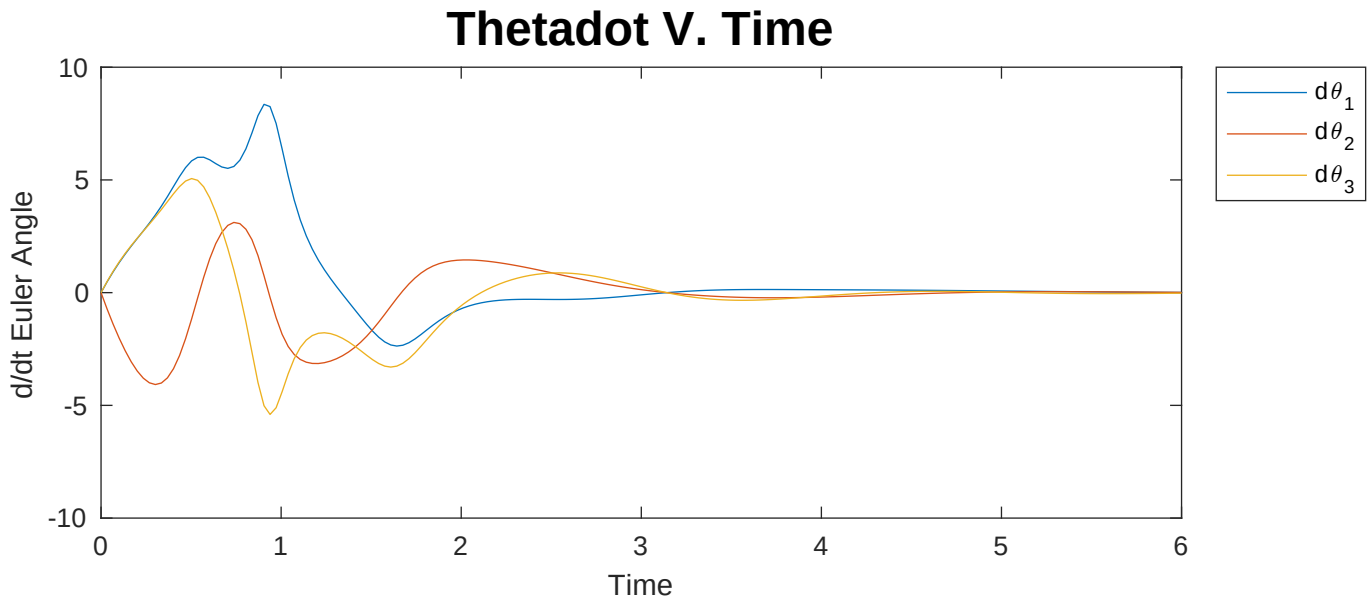
Finally, another important part of this code is plotting the rates of change as asked in part b. This was done as shown below in Figure 4.

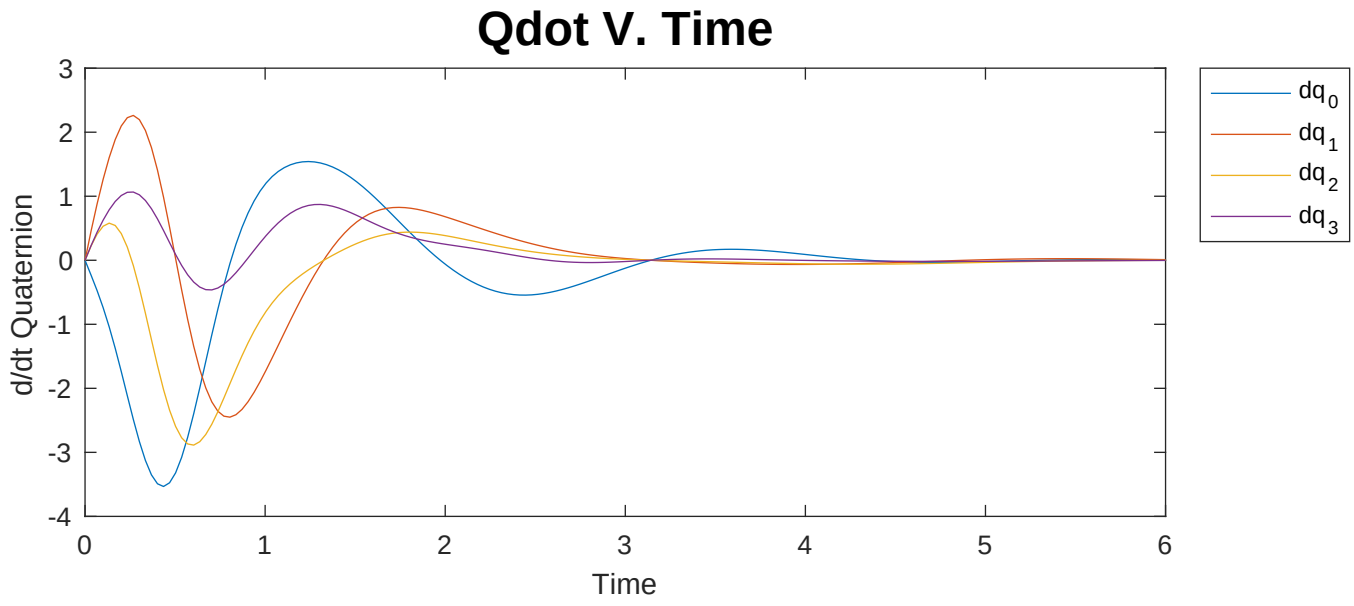
```
% FIND THE RATES
for i=1:length(t)
    rdot(:,i) = RtoX(getRdot(t(i),XtoR(xR(i,:)')));
    thetadot(:,i) = getHdot(t(i),xH(i,:)');
    [q0dot(i),qdot(:,i)] = getQdot(t(i),xQ(i,1),xQ(i,2:4)');
end

function [q0dot, qdot] = getQdot(t,q0,q)
% CODE TO COMPUTE Qdot = (q0dot,qdot)
omega = 10*exp(-t)*[sin(t);sin(2*t);sin(3*t)];
omegahat = [0 -omega(3) omega(2);...
            omega(3) 0 -omega(1);...
           -omega(2) omega(1) 0];

q0dot = -0.5*omega'*q;
qdot = 0.5*(omega*q0 - omegahat*q);

function thetadot = getHdot(t,theta)
% CODE TO COMPUTE Thetadot
theta2 = theta(2);
theta3 = theta(3);
omega = 10*exp(-t)*[sin(t);sin(2*t);sin(3*t)];

B=[(cos(theta3)/cos(theta2)) (-sin(theta3)/cos(theta2)) 0;...
    (sin(theta3)) (cos(theta3)) 0;...
    (-tan(theta2)*cos(theta3)) (tan(theta2)*sin(theta3)) 1];

thetadot = B*omega;

function Rdot = getRdot(t,R)
% CODE TO COMPUTE Rdot
omega = 10*exp(-t)*[sin(t);sin(2*t);sin(3*t)];
omegahat = [0 -omega(3) omega(2); omega(3) 0 -omega(1); -omega(2) omega(1) 0];

Rdot = R*omegahat;
```

Figure 4: Functions Rates of Change

Running the function, the plots on the next page are produced. Note how the rates of change converge to zero as the simulation approaches 6 seconds.


Rdot V. Time
d/dt Rotation Matrix
Time
dR1
dR2
dR3
dR4
dR5
dR6
dR7
dR8
dR9


Thetadot V. Time
d/dt Euler Angle
Time
dθ1
dθ2
dθ3


Qdot V. Time
d/dt Quaternion
Time
dq0
dq1
dq2
dq3