Table of Contents

0

```
close all;
clear all;
clc
addpath("C:\joshFunctionsMatlab\")
```

Problem 1

```
[Cx,Cy,Cz] = joshAxisRotation();

%a


r_1= [6738;3391;1953];
v_1=[-3.5;4.39;4.44];
zl_1 = r_1/norm(r_1);
yl_1 = cross(r_1,v_1)/norm(cross(r_1,v_1));
xl_1 = cross(yl_1,zl_1);
xe_1 = [1;0;0];
ye_1 = [0;1;0];
ze_1 = [0;0;1];

C21 = [[dot(xl_1,xe_1),dot(xl_1,ye_1),dot(xl_1,ze_1)];
[dot(yl_1,xe_1),dot(yl_1,ye_1),dot(yl_1,ze_1)];
[dot(zl_1,xe_1),dot(zl_1,ye_1),dot(zl_1,ze_1)]];
```

bcd

```
[a,phi] = joshRotM2PrincAxe(C21);

thetax = atan2(C21(2,3),C21(3,3));
thetay = asin(-C21(1,3));
thetaz = atan2(C21(1,2),C21(1,1));

C21verify = Cx(thetax)*Cy(thetay)*Cz(thetaz); % for my use
```

```
eta =@(C) .5*sqrt(1+trace(C));
epsilon =@(C,eta)...
    [(C(2,3)-C(3,2))/(4*eta);...
    (C(3,1)-C(1,3))/(4*eta);...
    (C(1,2)-C(2,1))/(4*eta)];
eta = eta(C21);
epsilon = epsilon(C21,eta);
disp("------------P1------------")
disp("My workings for this problem have the following results:")
disp("Rotation matrix from ECI to the spacecraft: ")
disp(round(C21,2))
disp("Principle axis (a) for this transformation: ")
disp(round(a,2))
disp("Principle angle of this transformation: "+ string(phi))
disp("Euler angles:")
disp("      thetax: " + string(round(rad2deg(thetax),2))+" degrees")
disp("      thetay: " + string(round(rad2deg(thetay),2))+" degrees")
disp("      thetaz: " + string(round(rad2deg(thetaz),2))+" degrees")
disp("quaternion:")
disp("      eta: "+string(round(eta,2)))
disp("      epsilon: ")
disp("      "+round(epsilon,2))
```

```
------------P1------------
My workings for this problem have the following results:
Rotation matrix from ECI to the spacecraft:
   -0.4900    0.6100    0.6200
    0.1200   -0.6600    0.7400
    0.8600    0.4400    0.2500

Principle axis (a) for this transformation:
    0.4900
    0.3900
    0.7800

Principle angle of this transformation: 2.8191
Euler angles:
      thetax: 71.36 degrees
      thetay: -38.35 degrees
      thetaz: 128.53 degrees
quaternion:
      eta: 0.16
      epsilon:
    "      0.48"
    "      0.38"
    "      0.77"
```

Problem 2

```
clear all;
[Cx,Cy,Cz] = joshAxisRotation();
```

```
syms t [3 1]

C21oft = Cz(t(3))*Cx(t(2))*Cz(t(1));

clear t t1 t2 t3

syms C [3 3]

t2 = acos(C(3,3)); %theta
t1 = asin(C(3,1)/sin(t2)); %phi
t3 = acos(C(1,1)/cos(t1)); %psi

disp("------------P2------------")
disp("My workings for this problem have the following results:")
disp("The formula for finding C21 in terms of phi = t1, theta = t2, and psi =
 t3, is:")
disp(C21oft)
disp("the formulas for finding the Euler angles are: ")
disp("t2 = "+string(t2))
disp("t1 = "+string(t1))
disp("t3 = "+string(t3))
disp("The singularities for this scheme are at t2 = 0 and t2 = pi. Physically
 we can consider a plane which rolls pitches and rolls again. If the plane
 ends its rotation pointing the oposite direction from where it began (a
 pitch of pi) or continues to point in the original direction (a pitch of
 0) then the last degree of freedom to define is the roll of the plane. This
 can be defined by either the first or second roll. How much the first vs
 second roll contributed cannot be determined from soley the initial and final
 orientations of the plane.")
```

```
------------P2------------
My workings for this problem have the following results:
The formula for finding C21 in terms of phi = t1, theta = t2, and psi = t3,
 is:
[  cos(t1)*cos(t3) - cos(t2)*sin(t1)*sin(t3), cos(t3)*sin(t1) +
 cos(t1)*cos(t2)*sin(t3), sin(t2)*sin(t3)]
[- cos(t1)*sin(t3) - cos(t2)*cos(t3)*sin(t1), cos(t1)*cos(t2)*cos(t3) -
 sin(t1)*sin(t3), cos(t3)*sin(t2)]
[                              sin(t1)*sin(t2),                              -
cos(t1)*sin(t2),         cos(t2)]

the formulas for finding the Euler angles are:
t2 = acos(C3_3)
t1 = asin(C3_1/(1 - C3_3^2)^(1/2))
t3 = acos(C1_1/(C3_1^2/(C3_3^2 - 1) + 1)^(1/2))
The singularities for this scheme are at t2 = 0 and t2 = pi. Physically we
 can consider a plane which rolls pitches and rolls again. If the plane ends
 its rotation pointing the oposite direction from where it began (a pitch
 of pi) or continues to point in the original direction (a pitch of 0) then
 the last degree of freedom to define is the roll of the plane. This can be
 defined by either the first or second roll. How much the first vs second
 roll contributed cannot be determined from soley the initial and final
 orientations of the plane.
```

Problem 3

```
clear all;
[Cx,Cy,Cz] = joshAxisRotation();

n= sqrt(2)/2
C21=[[n 0 -n];[0 1 0];[n 0 n]]
C32=[[0 0 -1];[-1 0 0];[0 1 0]]
clear n
```

```
n =

    0.7071


C21 =

    0.7071         0   -0.7071
         0    1.0000         0
    0.7071         0    0.7071


C32 =

     0     0    -1
    -1     0     0
     0     1     0
```

a

```
isRotM = @(M) (round(M*M',14) == eye(3) & round(M'*M,14) == eye(3) &
 round(det(M),14) == 1)

isRot21 = joshIsOnes(isRotM(C21));
isRot32 = joshIsOnes(isRotM(C32));
```

```
isRotM =

  function_handle with value:

    @(M)(round(M*M',14)==eye(3)&round(M'*M,14)==eye(3)&round(det(M),14)==1)
```

b

```
eta =@(C) .5*sqrt(1+trace(C));
epsilon =@(C)...
    [(C(2,3)-C(3,2))/(4*eta(C));...
```

```
    (C(3,1)-C(1,3))/(4*eta(C));...
    (C(1,2)-C(2,1))/(4*eta(C))];

C=@(eta,epsilon)
 (2*eta^2-1)*eye(3)+2*epsilon*epsilon'-2*eta*joshCross(epsilon)

eta21 = eta(C21);
epsilon21 = epsilon(C21);
eta32 = eta(C32);
epsilon32 = epsilon(C32);
C21verify = C(eta21,epsilon21);% for my use
C32verify = C(eta32,epsilon32);
```

```
C =

  function_handle with value:

    @(eta,epsilon)
(2*eta^2-1)*eye(3)+2*epsilon*epsilon'-2*eta*joshCross(epsilon)
```

c

```
eta31=eta21*eta32-epsilon32'*epsilon21;
epsilon31 = eta32*epsilon21+eta21*epsilon32+joshCross(epsilon21)*epsilon32;
```

de

```
C31 =C32*C21;
eta31verify = eta(C31);
epsilon31verify = epsilon(C31);
isSame1 = round(eta31verify,14) == round(eta31,14);
isSame2 = round(epsilon31verify,14) == round(epsilon31,14)
isSame2 = joshIsOnes(isSame2)
```

```
isSame2 =

  3×1 logical array

   1
   1
   1


isSame2 =

  logical

   1
```

f

```
eta21inv = eta(inv(C21));
epsilon21inv = epsilon(inv(C21));

isSame3 = round(eta21inv,14) == round(eta21,14);
isSame4 = round(epsilon21inv,14) == round(-epsilon21,14)
isSame4 = joshIsOnes(isSame4)

disp("------------P3------------")
disp("My workings for this problem have the following results:")
disp("C21 is a rotation matrix: "+string(isRot21))
disp("C32 is a rotation matrix: "+string(isRot32))
disp("quaternion21:")
disp("      eta: "+string(eta21))
disp("      epsilon: ")
disp("      "+epsilon21)
disp("quaternion32:")
disp("      eta: "+string(eta32))
disp("      epsilon: ")
disp("      "+epsilon32)
disp("quaternion31:")
disp("      eta: "+string(eta31))
disp("      epsilon: ")
disp("      "+epsilon31)
disp("C31:")
disp(string(C31))
disp("quaternion31 from C31:")
disp("      eta: "+string(eta31verify))
disp("      epsilon: ")
disp("      "+epsilon31verify)
disp("eta31 has been found both ways: "+string(isSame1))
disp("epsilon31 has been found both ways: "+string(isSame2))
disp("eta21 == eta21^-1: "+string(isSame3))
disp("-epsilon21 == epsilon21^-1: "+string(isSame4))
disp("The quaternions are related using:")
disp("-epsilon21 == epsilon21^-1, and eta21 == eta21^-1")
```

```
isSame4 =

  3×1 logical array

   1
   1
   1


isSame4 =

  logical

   1
```

```
------------P3------------
My workings for this problem have the following results:
C21 is a rotation matrix: true
C32 is a rotation matrix: true
quaternion21:
      eta: 0.92388
      epsilon:
    "      0"
    "      0.38268"
    "      0"

quaternion32:
      eta: 0.5
      epsilon:
    "      -0.5"
    "      0.5"
    "      0.5"

quaternion31:
      eta: 0.2706
      epsilon:
    "      -0.2706"
    "      0.65328"
    "      0.65328"

C31:
    "-0.70711"    "0"    "-0.70711"
    "-0.70711"    "0"    "0.70711"
    "0"           "1"    "0"

quaternion31 from C31:
      eta: 0.2706
      epsilon:
    "      -0.2706"
    "      0.65328"
    "      0.65328"

eta31 has been found both ways: true
epsilon31 has been found both ways: true
eta21 == eta21^-1: true
-epsilon21 == epsilon21^-1: true
The quaternions are related using:
-epsilon21 == epsilon21^-1, and eta21 == eta21^-1
```

Published with MATLAB® R2022a

```
function [a_v,phi] = joshRotM2PrincAxe(C)
arguments
    C (:,:) {mustBeReal, mustBeNumeric}
end

[m,n] = size(C);

if m ~= n % matrix must be square
    throw(MException("joshRotM2PrincAxe:invalidInput","Dimensions of C21 must
 match."))
end
clear m

if ~(round(C*C',14) == eye(n) & round(C'*C,14) == eye(n) & round(det(C),14) ==
 1) % checks that M is a rotation matrix, round so that an error near e-mach
 will not cause a failure
    throw(MException("joshPrincAxe:invalidInput","Matrix is not a rotation
 matrix."))
end
    phi = acos((trace(C)-1)/2); % calculates phi in terms of C
    if abs(phi - pi)<1e-14 % checks if there is a non unique solution
        warning("answer may not be unique")
    end
    a_v(1) = (C(2,3)-C(3,2))/(2*sin(phi)); % formula for components of a in
 terms of phi and C
    a_v(2) = (C(3,1)-C(1,3))/(2*sin(phi));
    a_v(3) = (C(1,2)-C(2,1))/(2*sin(phi));
    a_v = a_v';
end
```

Published with MATLAB® R2022a

```
function [isOnes] = joshIsOnes(M)
% takes a value (persumably a logical type matrix) and returns true iff all
% entries are true
[m,n] = size(M);
isOnes = true;
for i = 1:m
    for j = 1:n
        if M(i,j) ~= 1
            isOnes = false;
        end
    end
end
end
```

Published with MATLAB® R2022a

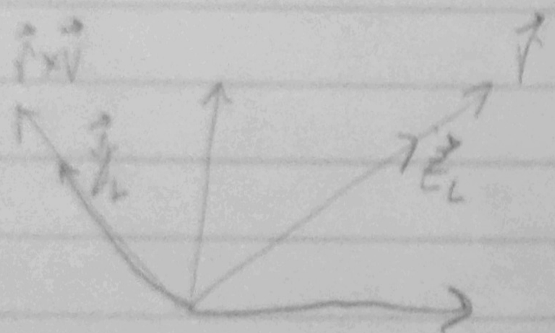
```
function mx = joshCross(m)
% takes a column vector and returns the associated 'cross' matrix such that
% mx*b == cross(m,b)
arguments
    m (3,1)
end
if isa(m,"double") % overloaded to handle either symbolic or double type
 vectors
    mx = zeros(3);
elseif isa(m,"sym")
    syms mx [3 3]
else
    throw(MException("joshCross:invalidInput","m must be type sym or double"))
end
    for i = 1:3
        mx(i,i) = 0;
    end
    mx(1,2) = -m(3);
    mx(1,3) =  m(2);
    mx(2,3) = -m(1);

    mx(2,1) =  m(3);
    mx(3,1) = -m(2);
    mx(3,2) =  m(1);
end
```

Published with MATLAB® R2022a

```
function [Cx,Cy,Cz] = joshAxisRotation(opt)
arguments
        opt {mustBeMember(opt,{'degree','radian'})} = 'radian'
end


if strcmp(opt,'degree')
Cx = @(theta)...
    [[1 0 0];...
    [0 cosd(theta) sind(theta)];...
    [0 -sind(theta) cosd(theta)]];

Cy = @(theta)...
    [[cosd(theta) 0 -sind(theta)];...
    [ 0 1 0];...
    [sind(theta) 0 cosd(theta)]];

Cz = @(theta)...
    [[cosd(theta) sind(theta) 0];...
    [-sind(theta) cosd(theta) 0];...
    [0 0 1]];
else
Cx = @(theta)...
    [[1 0 0];...
    [0 cos(theta) sin(theta)];...
    [0 -sin(theta) cos(theta)]];

Cy = @(theta)...
    [[cos(theta) 0 -sin(theta)];...
    [ 0 1 0];...
    [sin(theta) 0 cos(theta)]];

Cz = @(theta)...
    [[cos(theta) sin(theta) 0];...
    [-sin(theta) cos(theta) 0];...
    [0 0 1]];
end
end
```

Published with MATLAB® R2022a

HW 3



$$\vec r = \underline{\vec{\mathcal F}}_1^T \underline{r}_1 = \underline{\vec{\mathcal F}}_1^T \left(\; \right)$$

$$\vec z_L = \underline{\vec{\mathcal F}}_1^T \underline{z}_{L1} = -\frac{\vec r}{|\vec r|} = \underline{\vec{\mathcal F}}_1^T \frac{\underline{r}_1}{|\underline{r}_1|}$$

$$\underline{\vec{\mathcal F}}_1 = \begin{pmatrix} \vec x_E \\ \vec y_E \\ \vec z_E \end{pmatrix} = \underline{\vec{\mathcal F}}_1 \begin{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \\ \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \\ \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \end{pmatrix}$$