AAE 440: Spacecraft Attitude Dynamics

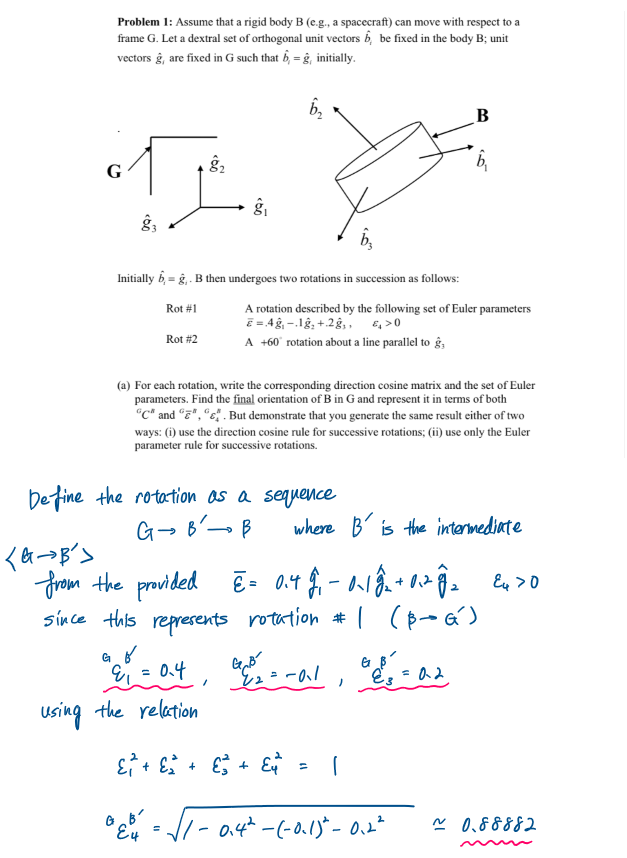
PS3: Successive Rotations and Orientational Angles

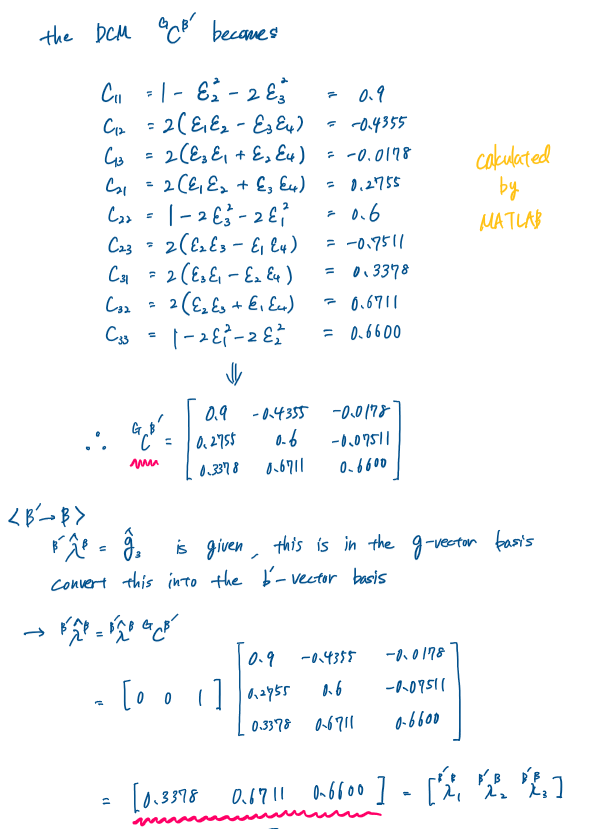
Tomoki Koike

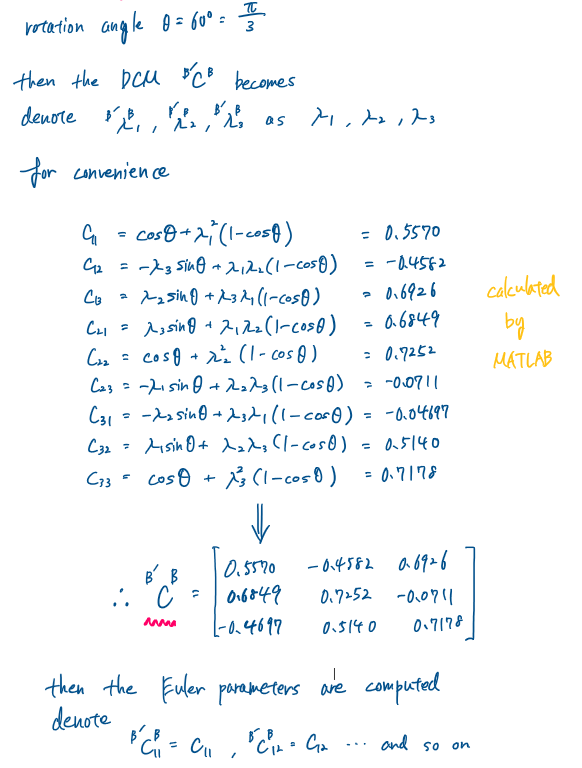
Friday February 7, 2020

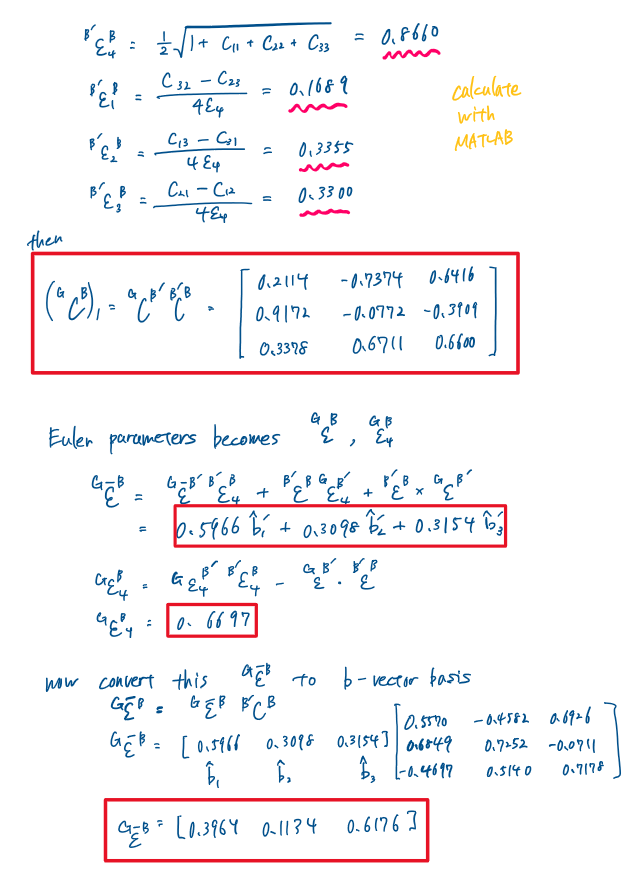
A close up of a tower

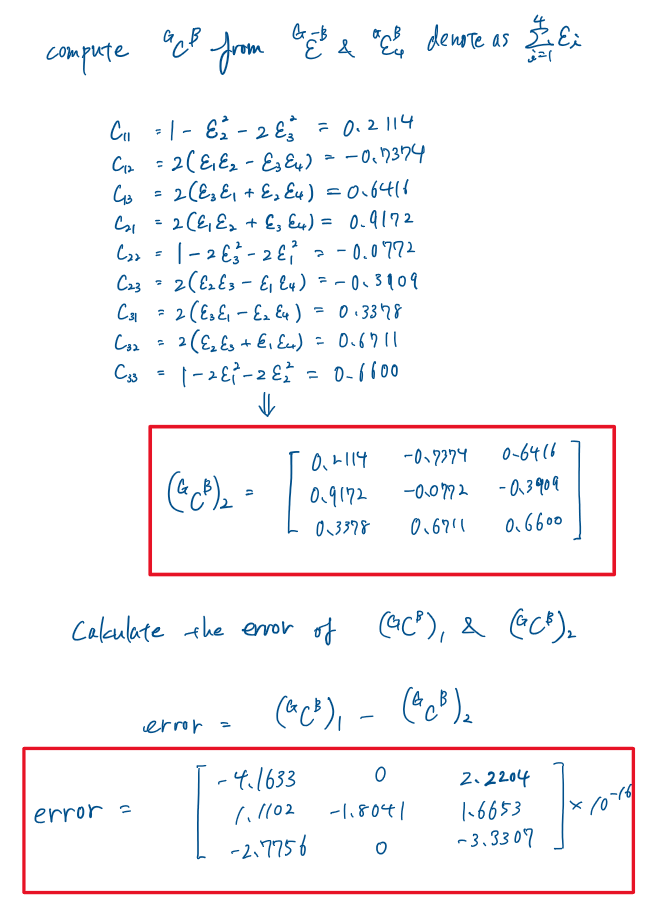
Description automatically generated

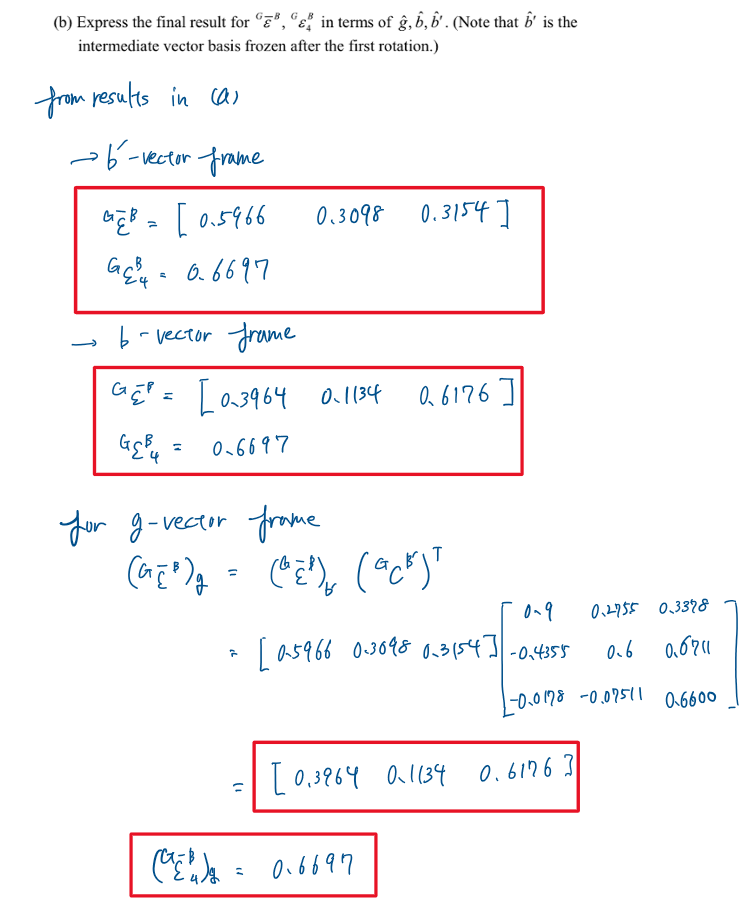


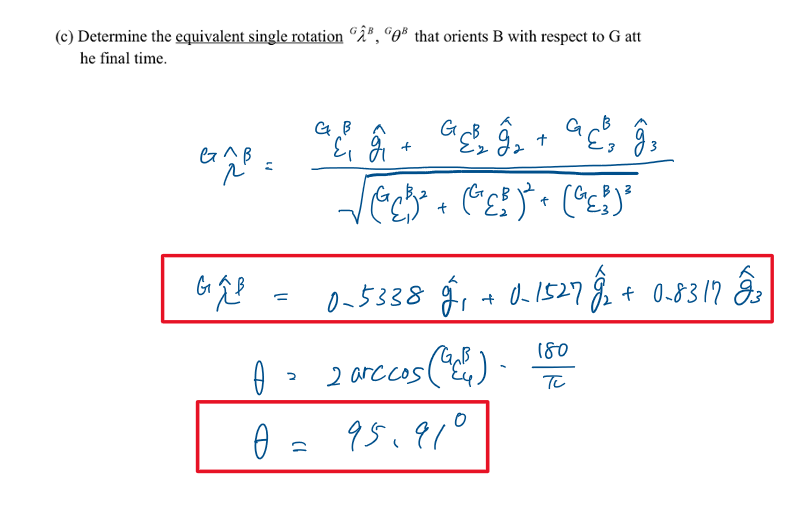


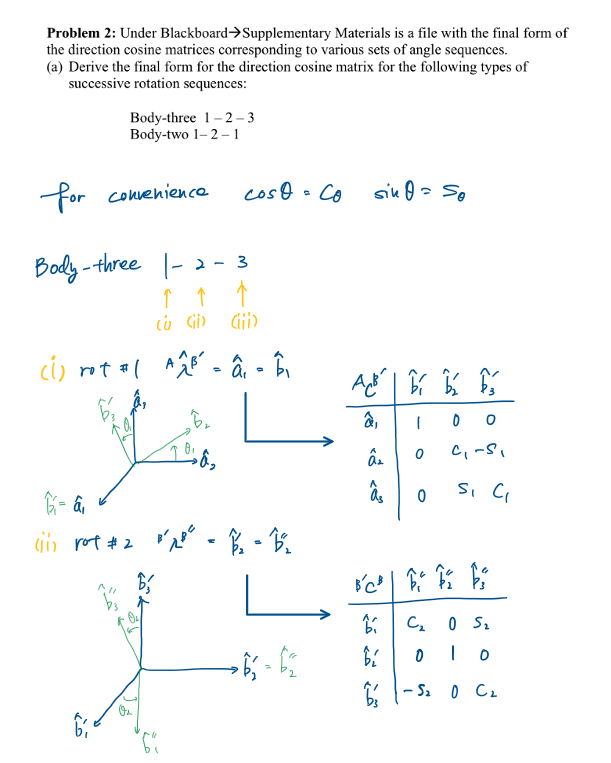


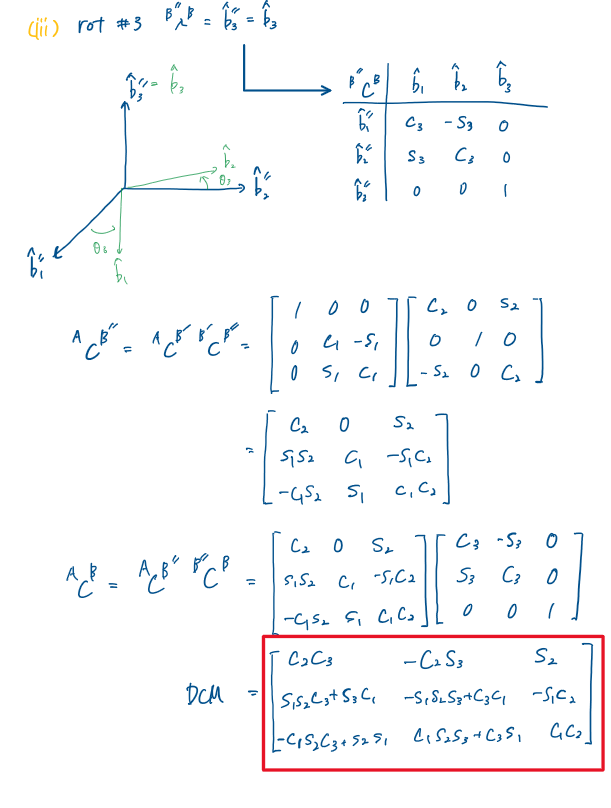


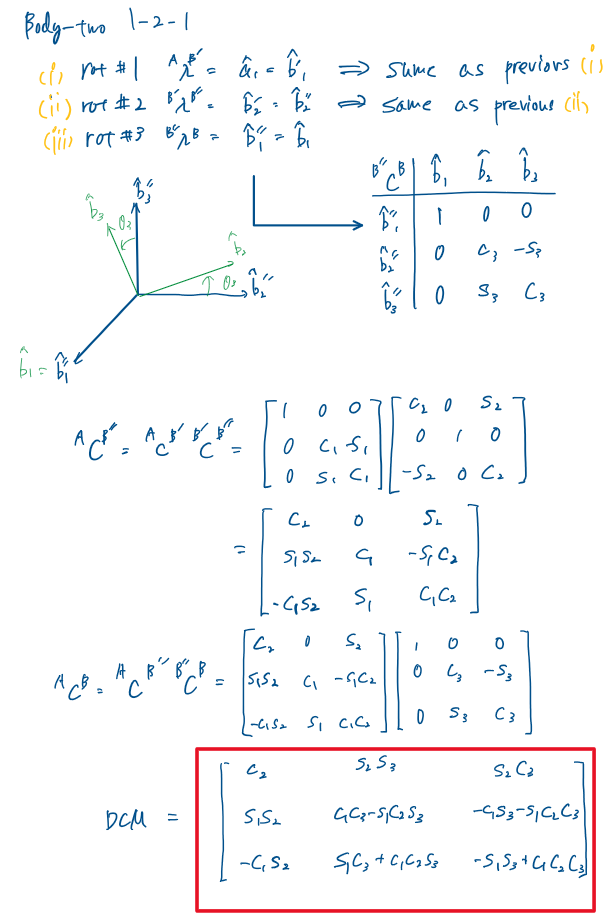


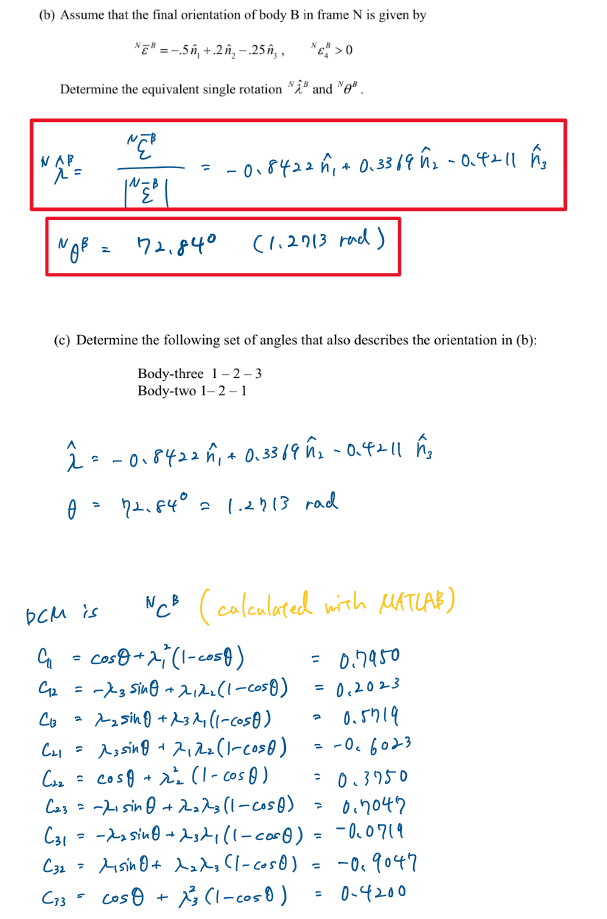


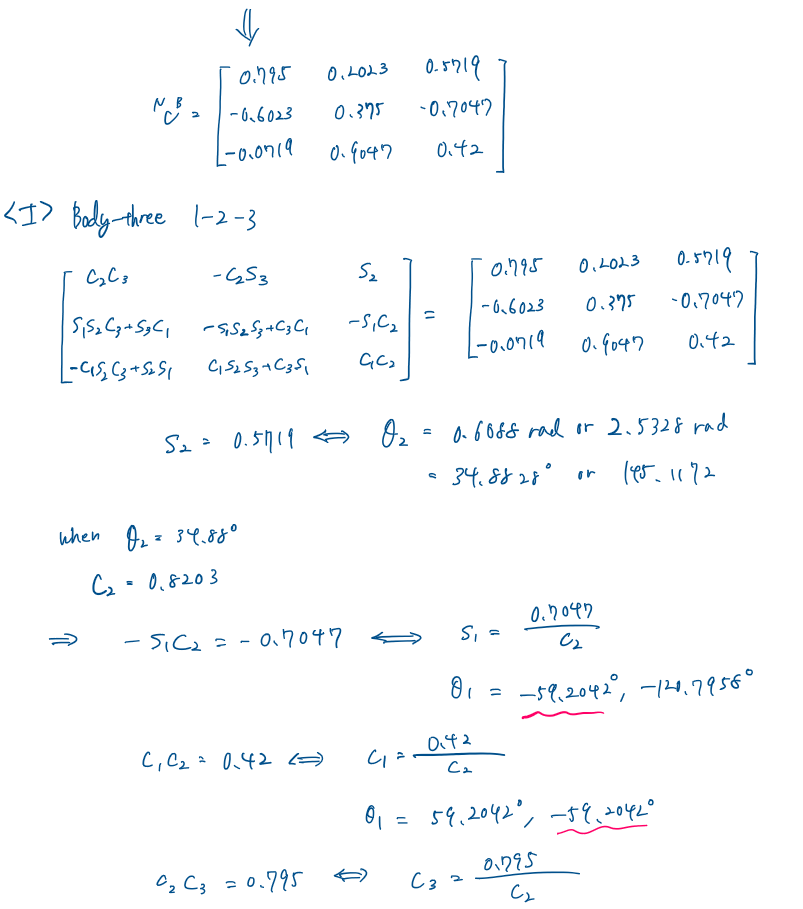


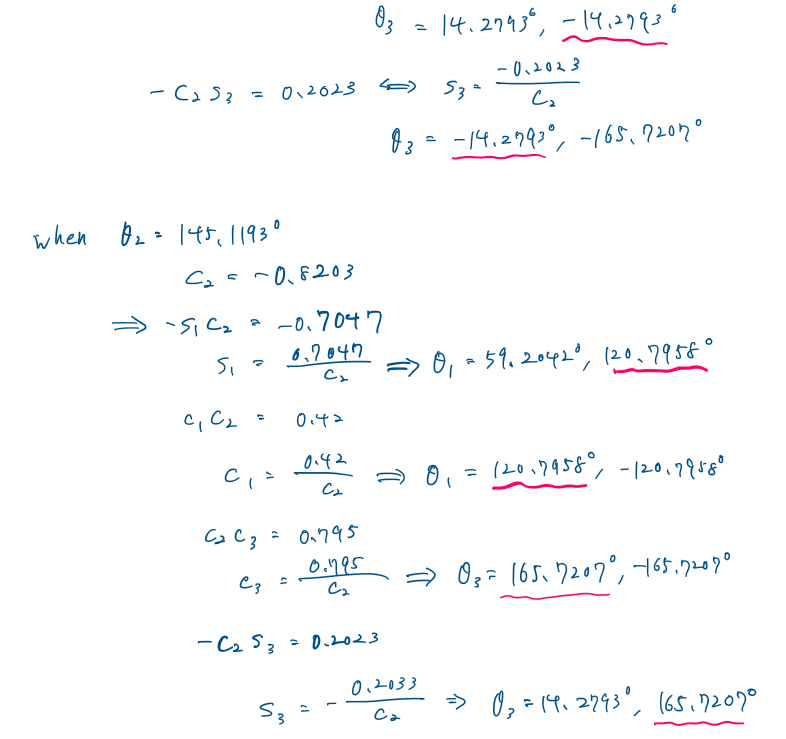


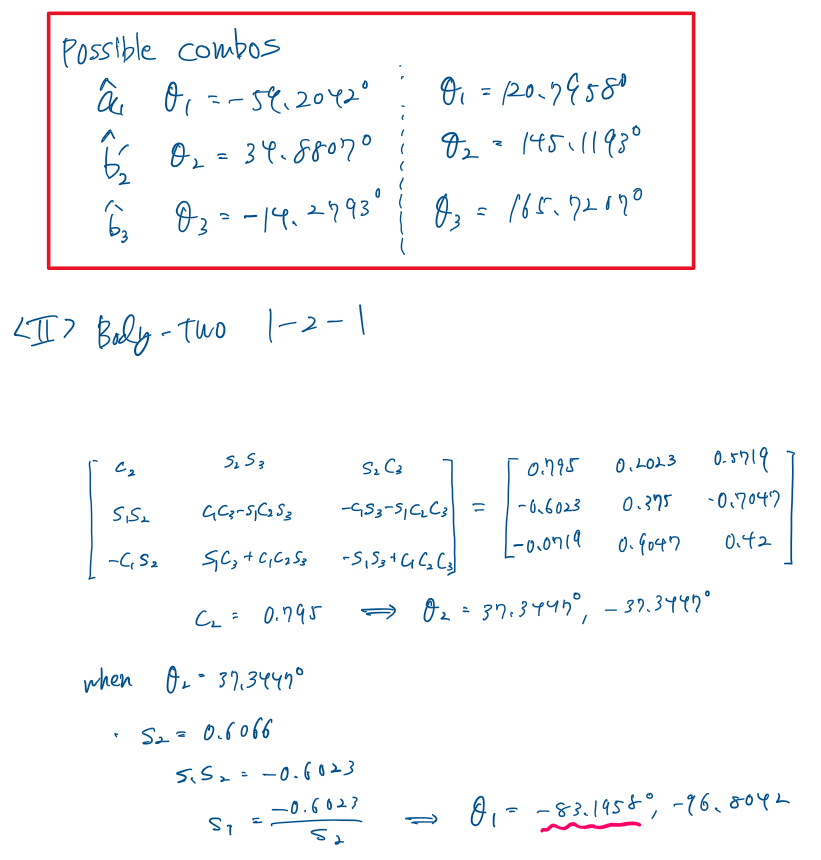


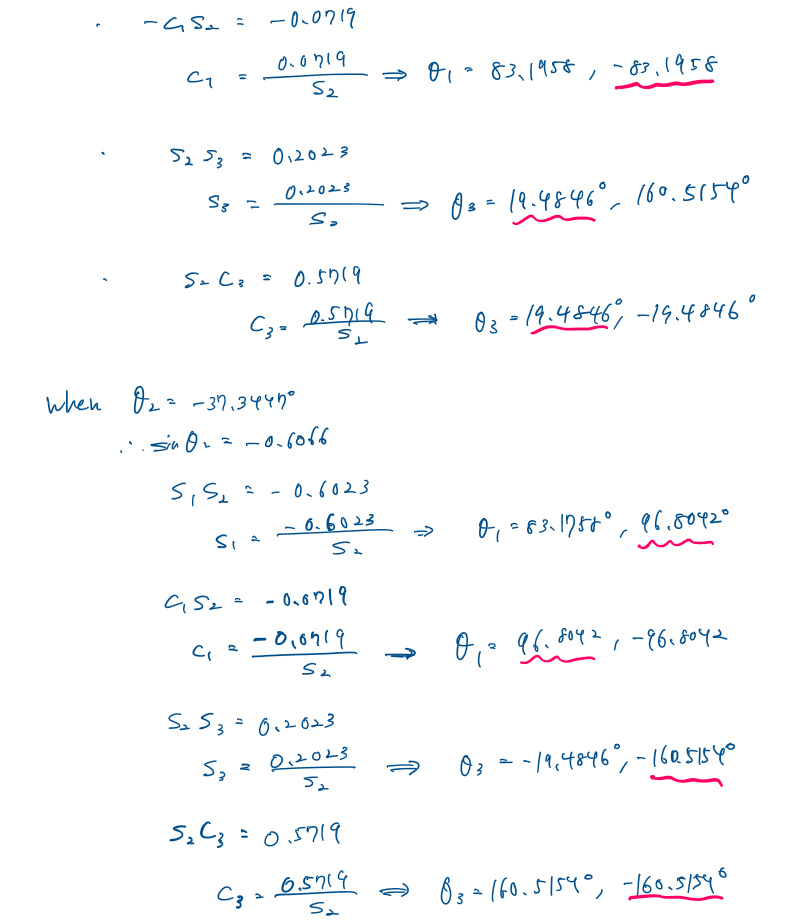


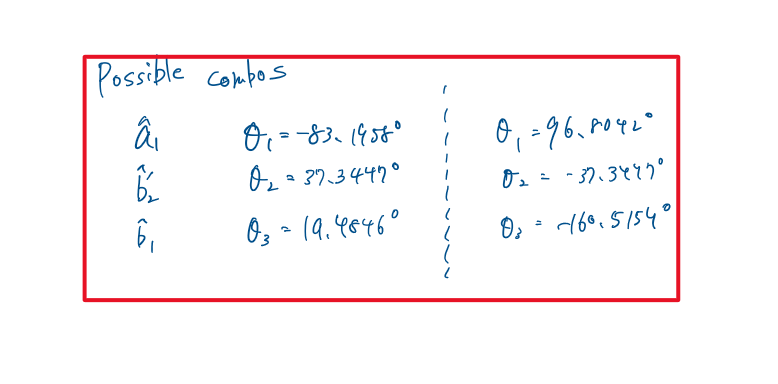


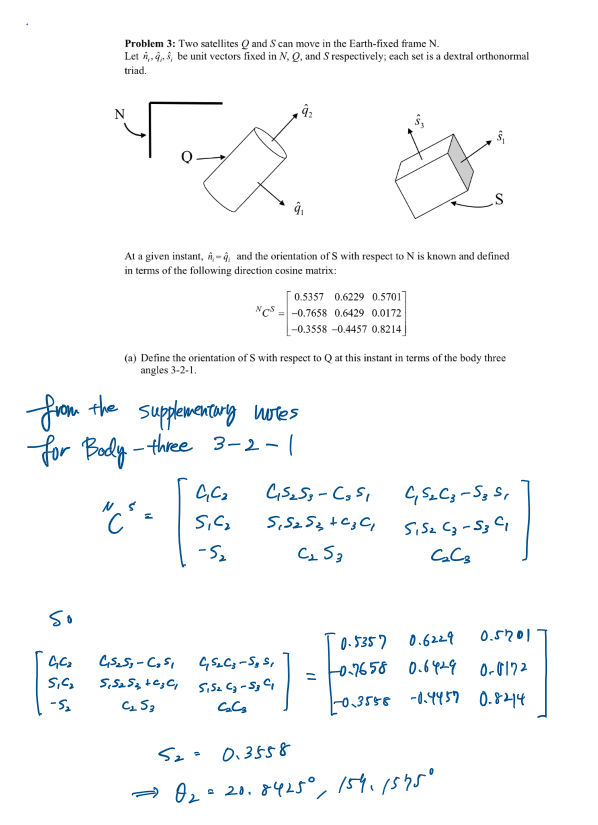


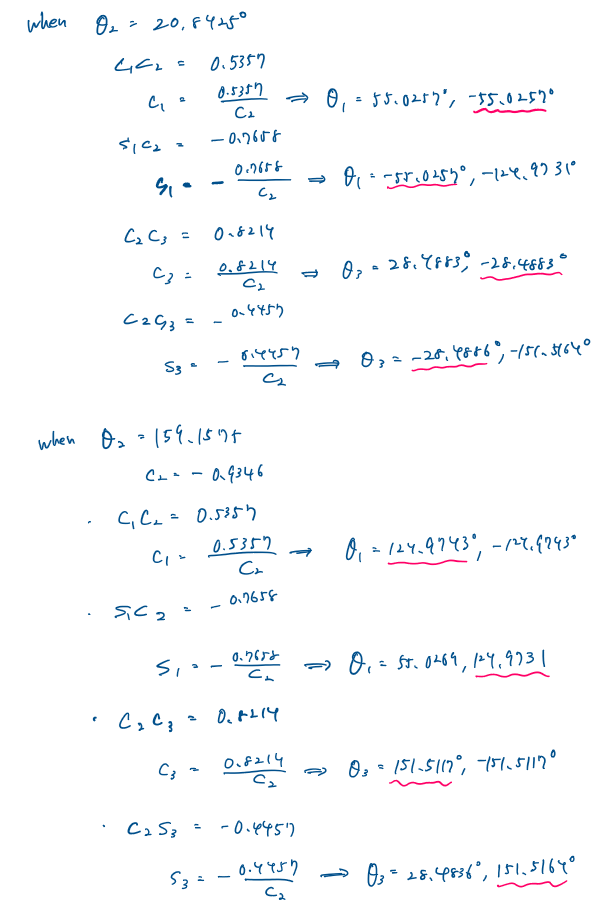


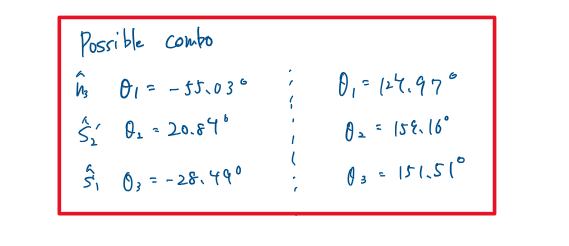


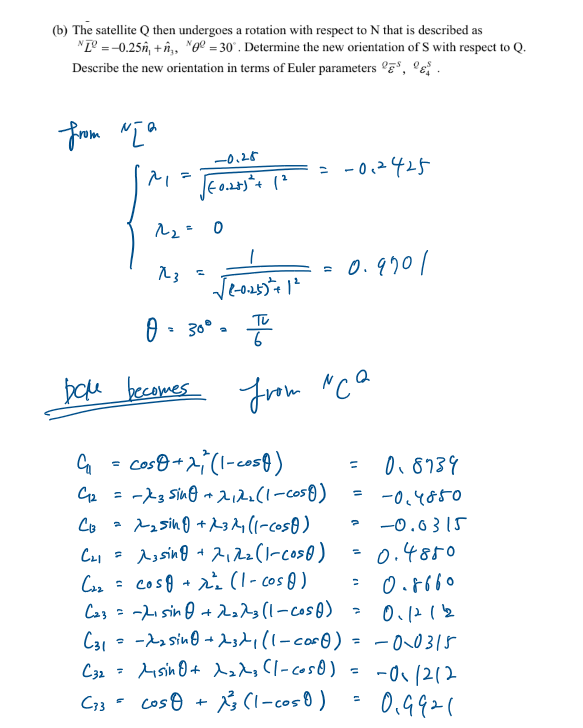


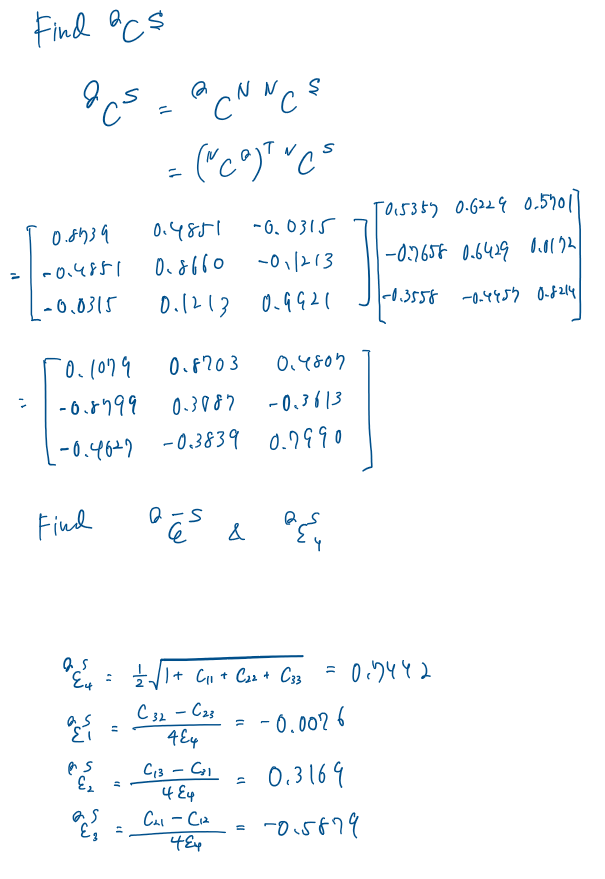


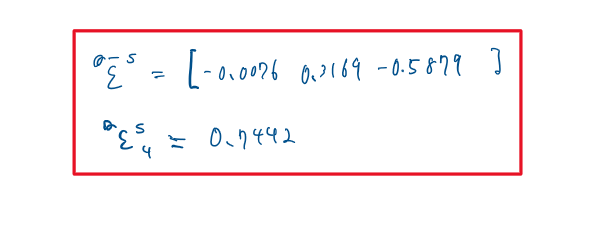












Appendix

AAE440 PS3 MATLAB CODE

**problem 1**

clear all; close all; clc;

**<a>**

% Rot#1 (G->B')

% Define the Euler Parameters/epsilons

e1\_1 = 0.4;

e1\_2 = -0.1;

e1\_3 = 0.2;

e1\_4 = sqrt(1-e1\_1^2-e1\_2^2-e1\_3^2);

% Set as vector

e1 = [e1\_1,e1\_2,e1\_3,e1\_4];

% Compute the DCM

C\_1 = DCM\_euler\_para(e1);

% Rot#2 (B'->B)

lambda\_g = [0 0 1]; % in g vector basis

lambda\_b\_prime = lambda\_g\*C\_1; % in b' vector basis

theta = pi/3; % rotation angle in radians

% Update DCM with next rotation

C\_2 = DCM\_lambda\_theta(lambda\_b\_prime,theta);

% Compute the Euler parameters

e2 = epsilons\_with\_DCM(C\_2);

% Compute the overall DCM

C = C\_1\*C\_2;

% Compute the successive Euler parameter

e\_new\_Bprime = eulerPara\_successive\_rot(e1, e2);

% Convert this to b-vector basis

e\_new\_vec = e\_new\_Bprime(1:3)\*C\_2;

e\_new\_B = [e\_new\_vec e\_new\_Bprime(4)];

C\_final = DCM\_euler\_para(e\_new\_B);

% Calculating the error

C\_error = C - C\_final;

**<b>**

% Epsilon in g frame

e\_g = e\_new\_Bprime(1:3)\*C\_1.';

**<c>**

% Compute the lambda and theta for one single rotation

e\_g = [e\_g e\_new\_Bprime(4)];

[lambda\_SRT, theta\_SRT] = lambda\_and\_theta\_fromEpsilon(e\_g);

% Covnert radians to degrees

theta\_SRT = theta\_SRT/pi\*180;

**problem 2**

clear all; close all; clc

**<b>**

e\_NB = [-0.5 0.2 -0.25];

e\_NB\_4 = sqrt(1 - sum(e\_NB.^2));

e\_NB = [e\_NB e\_NB\_4];

% Compute the lambda and theta for SRT

[lambda\_NB theta\_NB] = lambda\_and\_theta\_fromEpsilon(e\_NB);

% Convert lambda to degrees from radians

theta\_NB\_deg = theta\_NB\*180/pi;

**<b>**

% Calculating the DCM

C\_NB = DCM\_lambda\_theta(lambda\_NB, theta\_NB);

**problem 3**

**<b>**

clear all; close all; clc;

L = [-0.25 0 1];

theta = pi/6;

lambdas = zeros([1 3]);

lambdas(1) = L(1)/sqrt(L(1)^2+L(2)^2+L(3)^2);

lambdas(2) = L(2)/sqrt(L(1)^2+L(2)^2+L(3)^2);

lambdas(3) = L(3)/sqrt(L(1)^2+L(2)^2+L(3)^2);

% DCM from N basis to Q basis

C\_NQ = DCM\_lambda\_theta(lambdas, theta);

% DCM from N basis to S basis

C11 = 0.5357;

C12 = 0.6229;

C13 = 0.5701;

C21 = -0.7658;

C22 = 0.6429;

C23 = 0.0172;

C31 = -0.3558;

C32 = -0.4457;

C33 = 0.8214;

C\_NS = [C11 C12 C13; C21 C22 C23; C31 C32 C33];

% Compute the DCM from Q to S

C\_QS = C\_NQ.'\*C\_NS

% Compute epsilons for Q to S basis

epsilon = epsilons\_with\_DCM(C\_QS);

function C\_mat = DCM\_euler\_para(epsilons)

% Epsilon vector

epsilon1 = epsilons(1);

epsilon2 = epsilons(2);

epsilon3 = epsilons(3);

epsilon4 = epsilons(4);

% Calculating DCM with Euler parameters

C11 = 1 - 2\*epsilon2^2 - 2\*epsilon3^2;

C12 = 2\*(epsilon1\*epsilon2 - epsilon3\*epsilon4);

C13 = 2\*(epsilon3\*epsilon1 + epsilon2\*epsilon4);

C21 = 2\*(epsilon1\*epsilon2 + epsilon3\*epsilon4);

C22 = 1 - 2\*epsilon3^2 - 2\*epsilon1^2;

C23 = 2\*(epsilon2\*epsilon3 - epsilon1\*epsilon4);

C31 = 2\*(epsilon3\*epsilon1 - epsilon2\*epsilon4);

C32 = 2\*(epsilon2\*epsilon3 + epsilon1\*epsilon4);

C33 = 1 - 2\*epsilon1^2 - 2\*epsilon2^2;

C\_mat = [C11 C12 C13; C21 C22 C23; C31 C32 C33];

end

function C\_mat = DCM\_lambda\_theta(lambdas, theta)

% Lambda vector

lambda1 = lambdas(1);

lambda2 = lambdas(2);

lambda3 = lambdas(3);

% Calculating DCM with lambdas and theta

C11 = cos(theta) + lambda1^2\*(1-cos(theta));

C12 = -lambda3\*sin(theta) + lambda1\*lambda2\*(1-cos(theta));

C13 = lambda2\*sin(theta) + lambda3\*lambda1\*(1-cos(theta));

C21 = lambda3\*sin(theta) + lambda1\*lambda2\*(1-cos(theta));

C22 = cos(theta) + lambda2^2\*(1-cos(theta));

C23 = -lambda1\*sin(theta) + lambda2\*lambda3\*(1-cos(theta));

C31 = -lambda2\*sin(theta) + lambda3\*lambda1\*(1-cos(theta));

C32 = lambda1\*sin(theta) + lambda2\*lambda3\*(1-cos(theta));

C33 = cos(theta) + lambda3^2\*(1-cos(theta));

C\_mat = [C11 C12 C13; C21 C22 C23; C31 C32 C33];

end

function epsilons = epsilons\_with\_DCM(C\_mat)

epsilon4 = 0.5\*sqrt(1+C\_mat(1,1)+C\_mat(2,2)+C\_mat(3,3));

epsilon1 = (C\_mat(3,2)-C\_mat(2,3))/4/epsilon4;

epsilon2 = (C\_mat(1,3)-C\_mat(3,1))/4/epsilon4;

epsilon3 = (C\_mat(2,1)-C\_mat(1,2))/4/epsilon4;

epsilons = [epsilon1 epsilon2 epsilon3 epsilon4];

end

function e\_new = eulerPara\_successive\_rot(e1, e2)

e1\_v = e1(1:3);

e1\_4 = e1(4);

e2\_v = e2(1:3);

e2\_4 = e2(4);

% Calculate the successive epsilon

e\_v\_new = e1\_v\*e2\_4 + e2\_v\*e1\_4 + cross(e2\_v, e1\_v);

e4\_new = e1\_4 \* e2\_4 - dot(e1\_v,e2\_v);

e\_new = [e\_v\_new e4\_new];

end

function [lambda, theta] = lambda\_and\_theta\_fromEpsilon(epsilons)

% Calculating the lambda unit vector and the angle theta for a simple

% rotation using the epsilon values

e\_vec = epsilons(1:3);

e4 = epsilons(4);

% compute lambda

lambda = e\_vec / sqrt(sum(e\_vec.^2));

theta = 2\*acos(e4);

end