Project 5 :

Jack Chaille - jrc208 ; Justin Stanko - jts177 ; Krashagi Gupta - kxg360

Problem 1:

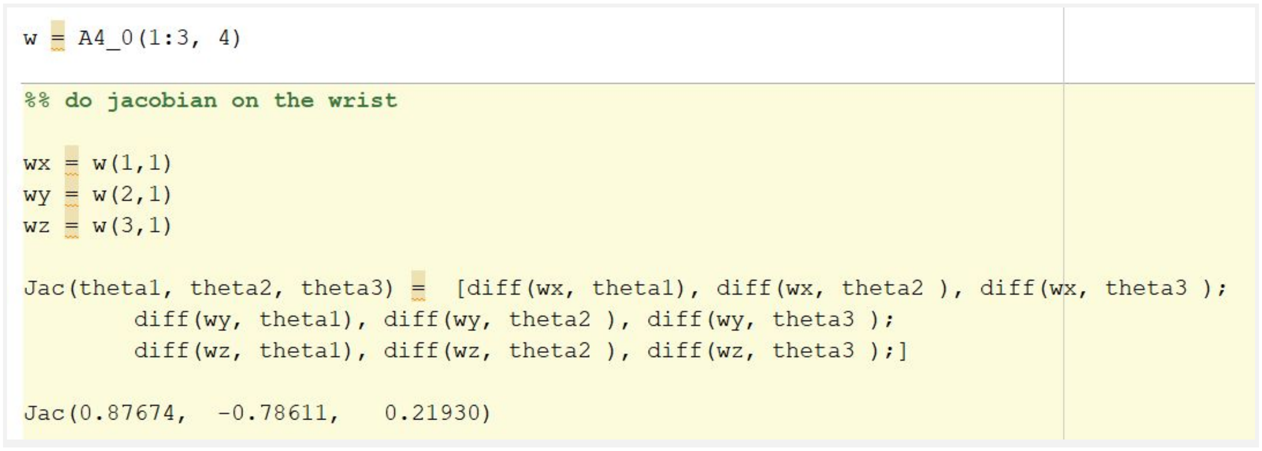
Compute an analytic expression for the position of the wrist point, w/0(q1,q2,q3). Compute an analytic expression for the 3x3 positional Jacobian of the wrist point, Jw(theta_vec).

Sol. I used the sym toolbox to do these calculations, after doing a little by hand. Using this formula, i computed A1_0, A2_1, A3_2 and A4_3:

$$\boldsymbol{A}_i^{i-1}(q_i) = \boldsymbol{A}_{i'}^{i-1}\boldsymbol{A}_i^{i'} = \begin{bmatrix} c_{\vartheta_i} & -s_{\vartheta_i}c_{\alpha_i} & s_{\vartheta_i}s_{\alpha_i} & a_i c_{\vartheta_i} \\ s_{\vartheta_i} & c_{\vartheta_i}c_{\alpha_i} & -c_{\vartheta_i}s_{\alpha_i} & a_i s_{\vartheta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}. \qquad (2.52)$$

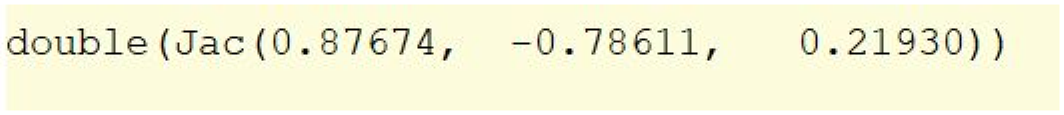
```
A1_0 = [cos(theta1), 0  , -sin(theta1),  a_vec(1)* cos(theta1) ;
        sin(theta1), 0  , cos(theta1),   a_vec(1)* sin(theta1) ;
        0          , -1 , 0          ,  d_vec(1)              ;
        0          , 0  ,   0         ,     1                  ;
        ];

A2_1 = [cos(theta2),  sin(theta2)  , 0 ,  a_vec(2)* cos(theta2) ;
        sin(theta2),  -cos(theta2) , 0  ,   a_vec(2)* sin(theta2) ;
        0          ,  0       ,     -1    ,  d_vec(2)              ;
        0          , 0       ,   0         ,     1                  ;
        ];

A3_2 = [cos(theta3),  0   , -sin(theta3) ,  a_vec(3)* cos(theta3) ;
        sin(theta3),    0  , cos(theta3),  a_vec(3)* sin(theta3) ;
        0          ,  -1       ,     0    ,  d_vec(3)              ;
        0          , 0       ,   0         ,     1                  ;
        ];

A4_3 = [cos(theta4), 0  , sin(theta4),   a_vec(4)* cos(theta4) ;
        sin(theta4), 0  , -cos(theta4),    a_vec(4)* sin(theta4) ;
        0          , 1  ,     0          ,  d_vec(4)              ;
        0          , 0  ,     0         ,     1                  ;
        ];


A4_0 = A1_0 * A2_1 * A3_2 * A4_3
```

Finding the wrist point from the vector A4_0 and partially differentiating it wrt theta1, theta2 and theta3.

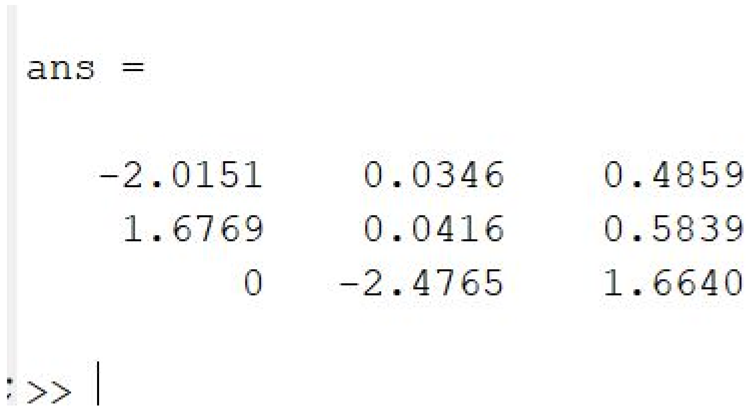


```
w = A4_0(1:3, 4)

%% do jacobian on the wrist

wx = w(1,1)
wy = w(2,1)
wz = w(3,1)

Jac(theta1, theta2, theta3) =  [diff(wx, theta1), diff(wx, theta2 ), diff(wx, theta3 );
        diff(wy, theta1), diff(wy, theta2 ), diff(wy, theta3 );
        diff(wz, theta1), diff(wz, theta2 ), diff(wz, theta3 );]

Jac(0.87674,  -0.78611,   0.21930)
```

The obtained Jacobian expression results was too convoluted to display here, but when evaluated the theta values as in the code, we get a result which is equal to the result found in the next section.



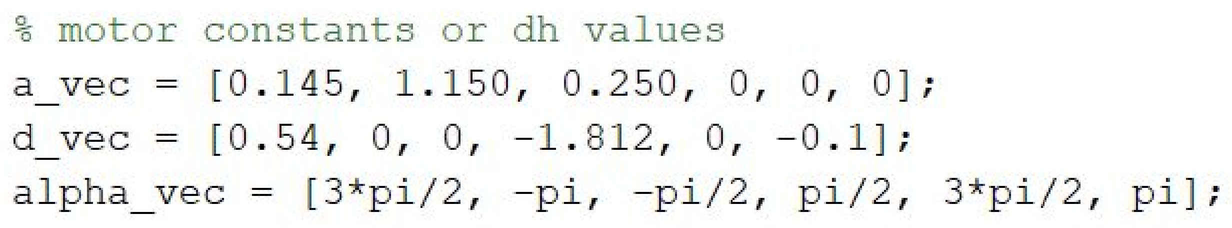
```
double(Jac(0.87674,  -0.78611,   0.21930))
```



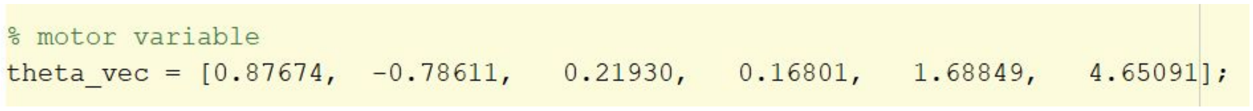
```
ans =

   -2.0151    0.0346    0.4859
    1.6769    0.0416    0.5839
         0   -2.4765    1.6640

>> |
```

Compose Matlab code to compute the wrist Jacobian using A matrices and vector cross products.

Compute A_ref, for the given robot and the given pose.
This is a variable that has all the
A vectors for the given robot i.e. A10, A20, A30…. A60

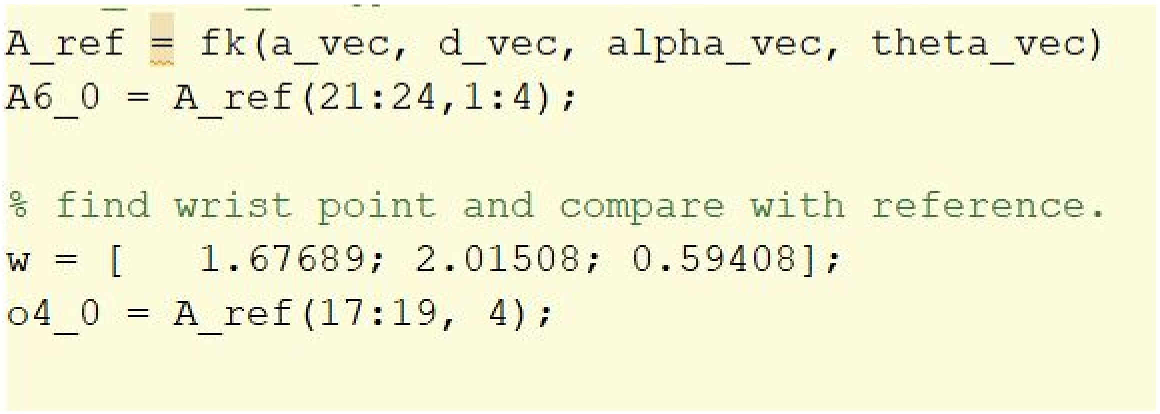


```
% motor constants or dh values
a_vec = [0.145, 1.150, 0.250, 0, 0, 0];
d_vec = [0.54, 0, 0, -1.812, 0, -0.1];
alpha_vec = [3*pi/2, -pi, -pi/2, pi/2, 3*pi/2, pi];
```



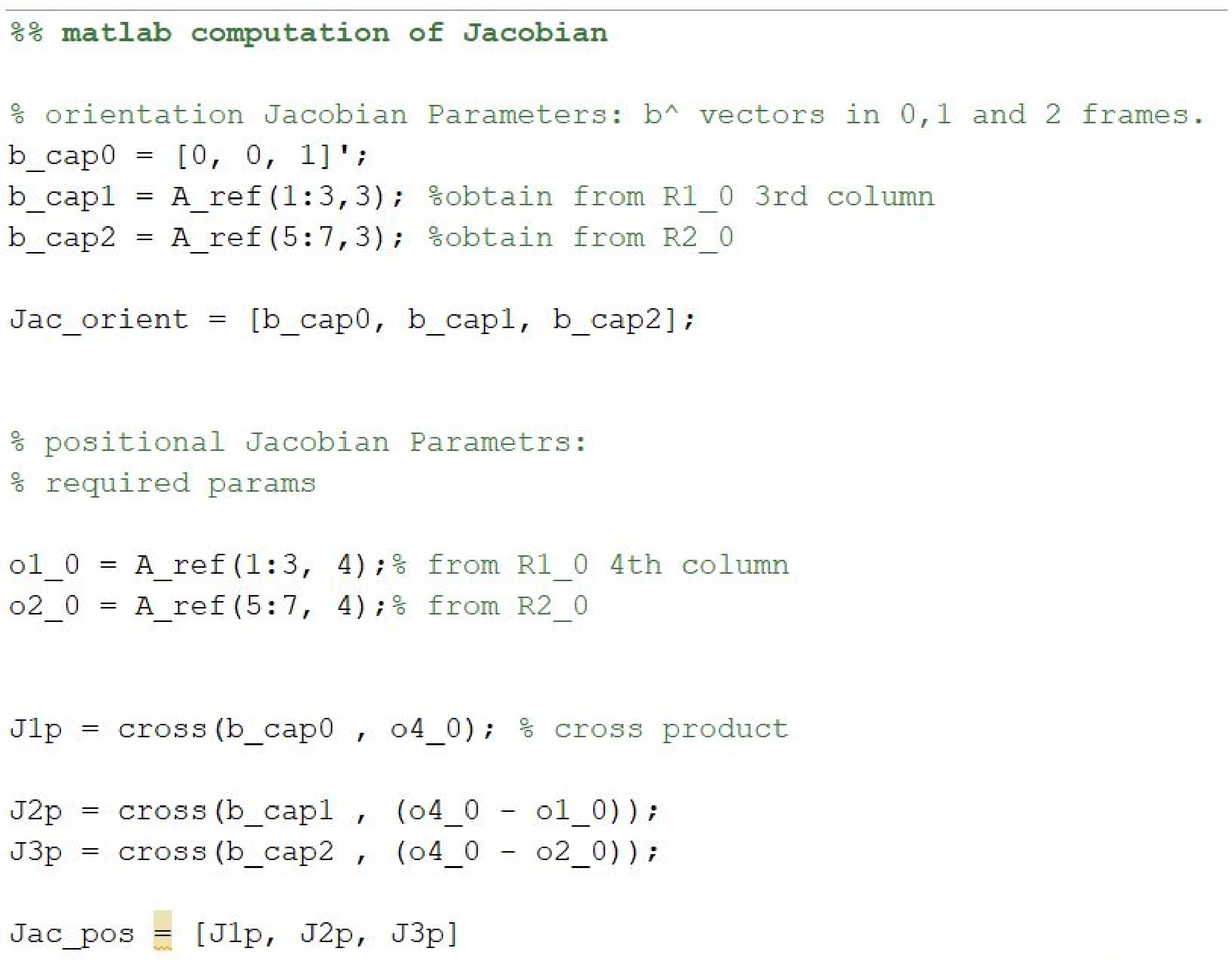
```
% motor variable
theta_vec = [0.87674,  -0.78611,   0.21930,   0.16801,   1.68849,   4.65091];
```

From this obtain A6_0 and wrist vector



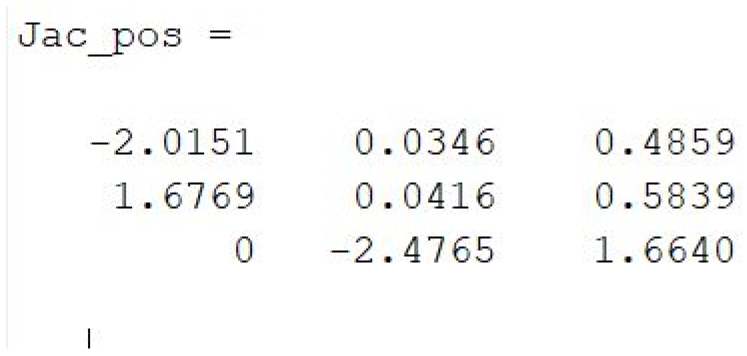
```
A_ref = fk(a_vec, d_vec, alpha_vec, theta_vec)
A6_0 = A_ref(21:24,1:4);

% find wrist point and compare with reference.
w = [   1.67689; 2.01508; 0.59408];
o4_0 = A_ref(17:19, 4);
```

First we find the orientational Jacobian parameters:
These are z vectors at the joints; obtained from A10 and A20,
And then we find distances between the joint to the end frames, we cross product it with
The z vectors obtained before this :



```
%% matlab computation of Jacobian

% orientation Jacobian Parameters: b^ vectors in 0,1 and 2 frames.
b_cap0 = [0, 0, 1]';
b_cap1 = A_ref(1:3,3); %obtain from R1_0 3rd column
b_cap2 = A_ref(5:7,3); %obtain from R2_0

Jac_orient = [b_cap0, b_cap1, b_cap2];


% positional Jacobian Parametrs:
% required params

o1_0 = A_ref(1:3, 4);% from R1_0 4th column
o2_0 = A_ref(5:7, 4);% from R2_0


J1p = cross(b_cap0 , o4_0); % cross product

J2p = cross(b_cap1 , (o4_0 - o1_0));
J3p = cross(b_cap2 , (o4_0 - o2_0));

Jac_pos = [J1p, J2p, J3p]
```

Result:



```
Jac_pos =

   -2.0151    0.0346    0.4859
    1.6769    0.0416    0.5839
         0   -2.4765    1.6640
```

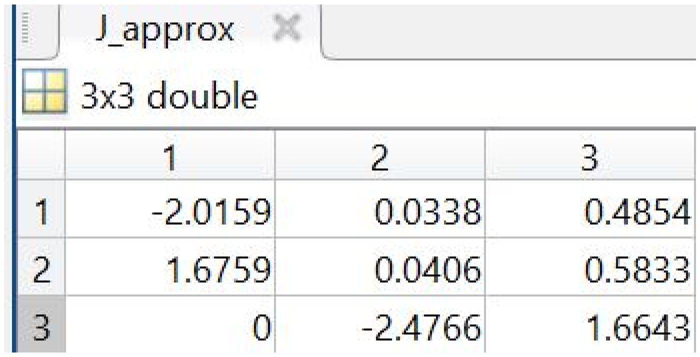
The result from both Analytical and matlab coding methods gives similar results.

Problem 2: %% 2

q_ref=[theta1,theta2,theta3];

q_dq1= q_ref+[0.001,0,0];

```
Aq1_1f0 = calculateA(0.145, 0.540, 3*pi/2, q_dq1(1));
Aq1_2f1 = calculateA(1.15, 0, -pi, q_dq1(2));
Aq1_3f2 = calculateA(0.25, 0, -pi/2, q_dq1(3));
Aq1_4f3 = calculateA(0, -1.812, pi/2, 0); %q4 doesn’t determine position of the wrist
Aq1_4f0=Aq1_1f0*Aq1_2f1*Aq1_3f2*Aq1_4f3;
w_dq1 = Aq1_4f0(1:3,4);

column1=((w_dq1-W)/(.001));

q_dq2= q_ref+[0,0.001,0];
Aq2_1f0 = calculateA(0.145, 0.540, 3*pi/2, q_dq2(1));
Aq2_2f1 = calculateA(1.15, 0, -pi, q_dq2(2));
Aq2_3f2 = calculateA(0.25, 0, -pi/2, q_dq2(3));
Aq2_4f3 = calculateA(0, -1.812, pi/2, 0);
Aq2_14f0=Aq2_1f0*Aq2_2f1*Aq2_3f2*Aq2_4f3;
w_dq2 = Aq2_14f0(1:3,4);

column2=((w_dq2-W)/(.001));

q_dq3= q_ref+[0,0,0.001];
Aq3_1f0 = calculateA(0.145, 0.540, 3*pi/2, q_dq3(1));
Aq3_2f1 = calculateA(1.15, 0, -pi, q_dq3(2));
Aq3_3f2 = calculateA(0.25, 0, -pi/2, q_dq3(3));
Aq3_4f3 = calculateA(0, -1.812, pi/2, 0);
Aq3_4f0=Aq3_1f0*Aq3_2f1*Aq3_3f2*Aq3_4f3;
w_dq3 = Aq3_4f0(1:3,4);

column3=((w_dq3-W)/(.001));

J_approx=[column1,column2,column3]

function A = calculateA(a,d, alpha, theta)
   A = eye(4, 4);
   A(1,1)= cos(theta);
   A(1,2)= -sin(theta)*cos(alpha);
   A(1,3)= sin(theta)*sin(alpha);
   A(1,4)= a*cos(theta);
   A(2,1)= sin(theta);
   A(2,2)= cos(theta)*cos(alpha);
   A(2,3)= -cos(theta)*sin(alpha);
   A(2,4)= a*sin(theta);
   A(3,1)= 0;
   A(3,2)= sin(alpha);
   A(3,3)= cos(alpha);
   A(3,4)= d;
   A(4,1)= 0;
   A(4,2)= 0;
   A(4,3)= 0;
   A(4,4)= 1;
end
```



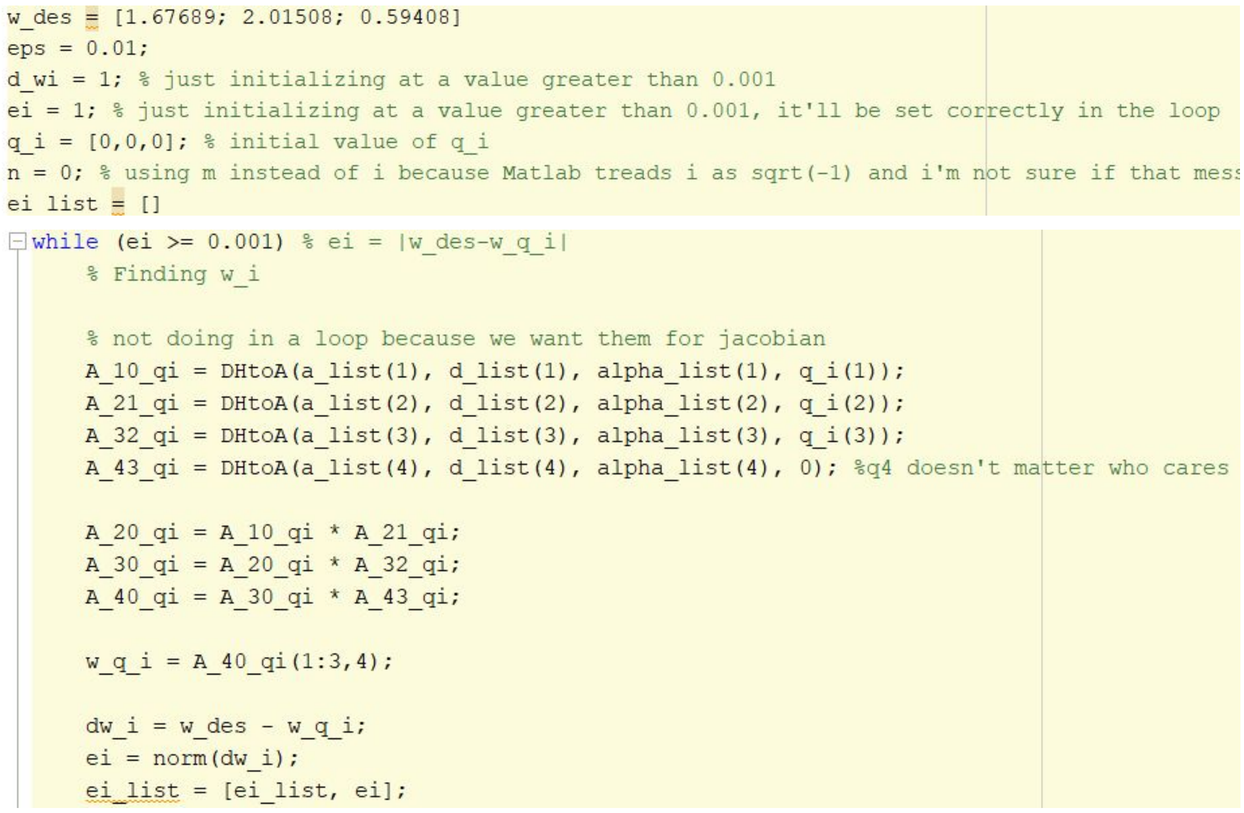
J_approx

3x3 double

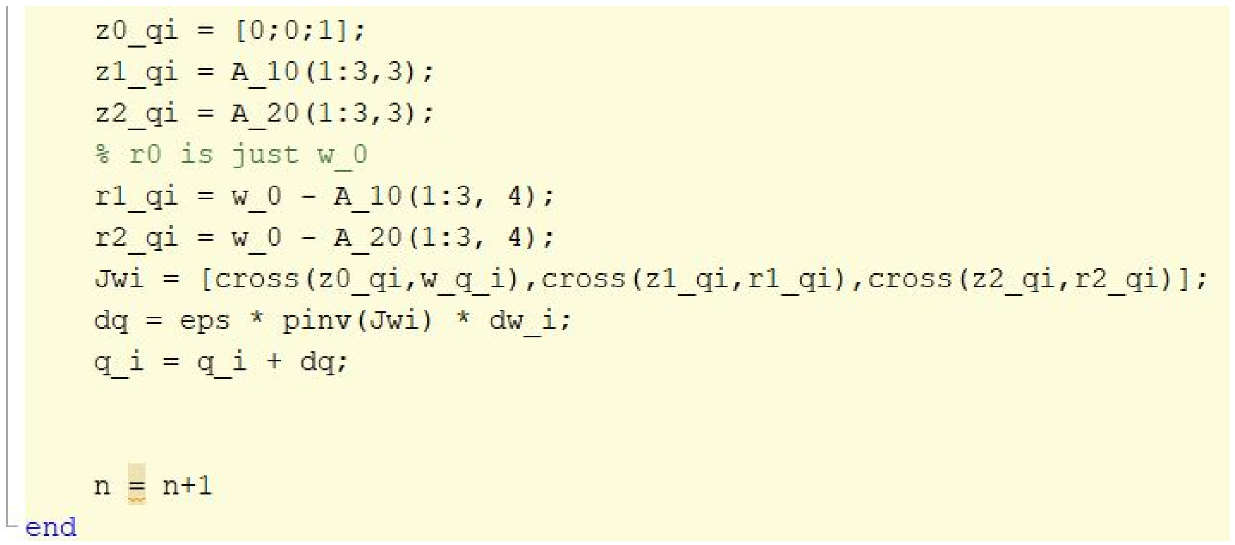
	1	2	3
1	-2.0159	0.0338	0.4854
2	1.6759	0.0406	0.5833
3	0	-2.4766	1.6643

This value for the Jacobian is slightly off from our initial computation of the Jacobian in problem 1. This is due to our step size of 0.001. The jacobian calculated in problem one is the limit of the function while problem 2 we take an approximation at increments of 0.001. The approximation will get closer to the actual value in 1 as step size decreases and the two would be equal with an infinitely small step size.

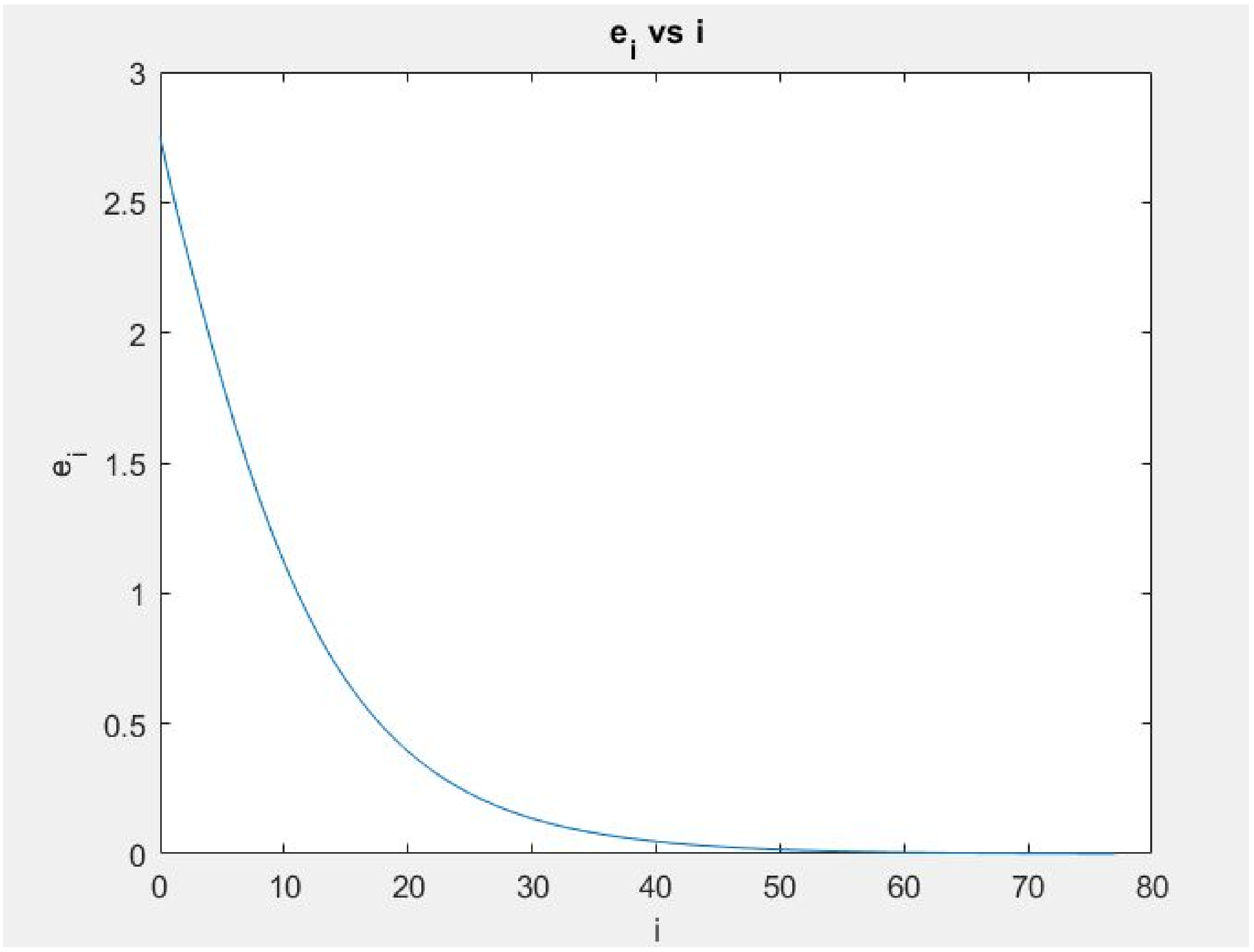
Problem 3:



```
w_des = [1.67689; 2.01508; 0.59408]
eps = 0.01;
d_wi = 1; % just initializing at a value greater than 0.001
ei = 1; % just initializing at a value greater than 0.001, it'll be set correctly in the loop
q_i = [0,0,0]; % initial value of q_i
n = 0; % using m instead of i because Matlab treads i as sqrt(-1) and i'm not sure if that mess
ei_list = []
while (ei >= 0.001) % ei = |w_des-w_q_i|
    % Finding w_i

    % not doing in a loop because we want them for jacobian
    A_10_qi = DHtoA(a_list(1), d_list(1), alpha_list(1), q_i(1));
    A_21_qi = DHtoA(a_list(2), d_list(2), alpha_list(2), q_i(2));
    A_32_qi = DHtoA(a_list(3), d_list(3), alpha_list(3), q_i(3));
    A_43_qi = DHtoA(a_list(4), d_list(4), alpha_list(4), 0); %q4 doesn't matter who cares

    A_20_qi = A_10_qi * A_21_qi;
    A_30_qi = A_20_qi * A_32_qi;
    A_40_qi = A_30_qi * A_43_qi;

    w_q_i = A_40_qi(1:3,4);

    dw_i = w_des - w_q_i;
    ei = norm(dw_i);
    ei_list = [ei_list, ei];
```



```
    z0_qi = [0;0;1];
    z1_qi = A_10(1:3,3);
    z2_qi = A_20(1:3,3);
    % r0 is just w_0
    r1_qi = w_0 - A_10(1:3, 4);
    r2_qi = w_0 - A_20(1:3, 4);
    Jwi = [cross(z0_qi,w_q_i),cross(z1_qi,r1_qi),cross(z2_qi,r2_qi)];
    dq = eps * pinv(Jwi) * dw_i;
    q_i = q_i + dq;


    n = n+1
end
```



A value of epsilon = 0.01 was chosen as the e_i vs i plot converged quickly but smoothly, so in a real application there wouldn’t be the risk of overshooting (if epsilon was too large) or taking exceedingly long (if epsilon was too small).

Challenge Problem:

In the textbook, there are three ways by which this problem can be solved.
What concerns the orientation error, its expression
depends on the particular representation of end-effector orientation, namely,
Euler angles, angle and axis, and unit quaternion.

Euler angle :

$$\boldsymbol{e}_O = \boldsymbol{\phi}_d - \boldsymbol{\phi}_e(\boldsymbol{q})$$

Orientation error is calculated using euler angle desired and euler angle as a function of q

Time derivative :

$$\dot{\boldsymbol{e}}_O = \dot{\boldsymbol{\phi}}_d - \dot{\boldsymbol{\phi}}_e.$$

the Jacobian inverse solution for a nonredundant manipulator is derived
From

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_A^{-1}(\boldsymbol{q}) \begin{bmatrix} \dot{\boldsymbol{p}}_d + \boldsymbol{K}_P \boldsymbol{e}_P \\ \dot{\boldsymbol{\phi}}_d + \boldsymbol{K}_O \boldsymbol{e}_O \end{bmatrix}$$

where KP and KO are positive definite matrices.