ECSE/CSDS/EMAE 489
Problem Set 5: Jacobians
Assigned: 3/3/21
Due: 3/10/21

1) Jacobian Computation

For the same Motoman robot as in PS3 and PS4, we will want to compute the Jacobian. First make sure your forward kinematics is correct. You should confirm you get the following:

at theta_vec (DH theta values) = [0.87674, -0.78611, 0.21930, 0.16801, 1.68849, 4.65091],
The toolflange origin is at:
p_tool = [1.69134; 2.05840; 0.68304]

The toolflange orientation (R_6/0) is:
R_tool =

```
  0.84807   0.50982   0.14446
 -0.51742   0.73797   0.43321
  0.11425  -0.44214   0.88964
```

and the wrist point at [theta1,theta2,theta3] = [0.87674, -0.78611, 0.21930] is:
w = [1.67689; 2.01508; 0.59408]

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, **J**w(theta_vec).

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

Evaluate your Jacobians for [theta1,theta2,theta3] = [0.87674, -0.78611, 0.21930]. Prove that your results (for both methods) is:
Jw(theta_vec) =

```
 -2.01508   0.03459   0.48593
  1.67689   0.04157   0.58393
  0.00000  -2.47655   1.66395
```

2) Jacobian numerical validation

Use the above wrist point **w** For **q**_ref = [theta1,theta2,theta3] = [0.87674, -0.78611, 0.21930].

Consider perturbations in joint space for the first three joint angles as follows.

Compute the wrist point for joint angles **q**_dq1 = **q**_ref +[0.001,0,0];
Call the resulting wrist pose: **w**_dq1.
Compute the vector (**w**_dq1-**w**)/(0.001).
Install this as the first column of a 3x3 matrix called: **J**_approx.

Compute the wrist point for joint angles **q**_dq2 = **q**_ref +[0, 0.001,0];
Call this **w**_dq2.

Compute the vector (**w**_dq2-**w**)/(0.001).
Install this as the second column of **J**_approx.

Compute the wrist point for joint angles **q**_dq3 = **q**_ref +[0,0,0.001];
Call this **w**_dq3.
Compute the vector (**w**_dq3-**w**)/(0.001).
Install this as the third column of **J**_approx.

Comment on how this compares to your computed **J**w(**q**_ref) and why.

3) Jacobians for numerical inverse kinematics
Use the following algorithm to solve inverse kinematics for wrist pose
    w_des = [1.67689; 2.01508; 0.59408].

* define d**w**_i = **w**_des-**w**(**q**_i)
* define **J**wi = **J**w(**q**_i) where **J**w is the wrist-point positional Jacobian
* define eps_i for each iteration (how to choose this is up to you)

Iterate as follows until ||d**w**_i||< 0.001m
initialize i=0 and **q**_0 = [0;0;0]
- compute d**w**_i = **w**_des-**w**(**q**_i)
-compute ei = ||**w**_des-**w**(**q**_i)|| (test if this meets the termination condition)
-compute **J**wi= **J**w(**q**_i) (3x3)
-compute d**q** = eps_i*inverse(**J**wi)*d**w**_i
-update **q**_(i+1) = **q**_i + d**q**
-repeat
Plot out e_i vs iteration number, I
Describe your means for choosing eps_i and how you arrived at this choice.
Comment on your results

Challenge question:
Describe how to extend Q3 to include orientation for a full 6DOF solution. How do you treat orientation error? More ambitiously, implement your method and document your result. Plot error vs. iteration number.

Deliverables:
Include a description of your approach to your solutions. Include your code (presumably in Matlab). Include your plots. All members of a group will submit the same solution (redundantly). Don’t forget to enter your group/self rankings with your submission.