```matlab
close all
clear
clc

% The test section that validates two functions.
% ------------test starts------------
Links = [0.07; 0.03; 0.02]; % assign links
configuration = [pi/6; pi/8; pi/7]; % assign the input configuration
pose = RRR_direct_2D(Links, configuration); % calculate pose
% display the pose calculated in function RRR_direct_2D
disp("test for direct kinematics(phi in angles, not radians):")
disp(pose);
% input the pose calculated above into function RRR_inverse_2D to check if it matches the input configuration
config = RRR_inverse_2D(Links, pose);
% display the output configuration and check if it matches the input
% config.
disp("test for inverse kinematics, with 2 sets of solutions(elbow up and elbow down)");
disp(config);
% ------------test ends--------------




%-------------problem (d)------------
numberOfConfigs = 5; % assign #.of valid configs. we need 5 as stated in problem d.
xForce = 0.5;  % assign x-component force
yForce = 0.05; % assign y-component force
forceVector = [xForce; yForce]; % assign to a vector
% assign link lengths
l1 = 0.04;
l2 = 0.03;
l3 = 0.02;
links = [l1;l2;l3];
phi = atan2(yForce,xForce); % compute the angle for the end effector
% compute 5 random configurations for three joints and display
configs_d = RRR_random_config(numberOfConfigs,phi);
disp("5 random configurations:");
disp(configs_d);
% compute 5 set of torques corresponding to 5 random configurations and display
torques = RRR_torque_2D(forceVector,numberOfConfigs,configs_d,links);
disp("5 sets of torque corresponding to random configs:")
disp(torques);

%-------------end of problem (d)-----







function pose_end = RRR_direct_2D(Link_Lengths, config) % question (a)
angleConverter = 180/pi;  % convert radian to angles
% calculate x using D.P.K equations
pose_end(1) = Link_Lengths(1)*cos(config(1))+Link_Lengths(2)*cos(config(1)+config(2))+Link_Lengths(3)*cos(config(1)+config(2)+config(3));
% calculate y using D.P.K equations
pose_end(2) = Link_Lengths(1)*sin(config(1))+Link_Lengths(2)*sin(config(1)+config(2))+Link_Lengths(3)*sin(config(1)+config(2)+config(3));
% calculate phi using D.P.K equations
pose_end(3) = (config(1) + config(2) + config(3))*angleConverter;
end

function config = RRR_inverse_2D(Link_Lengths, pose_end)  % question (b)
radianConverter = pi/180; %convert angles to radian;
x = pose_end(1); % assign x
y = pose_end(2); % assign y
phi = pose_end(3)*radianConverter;  % assign phi
l1 = Link_Lengths(1); % assign link 1
l2 = Link_Lengths(2); % assign link 2
l3 = Link_Lengths(3); % assign link 3
% calculate cosine of theta2 using I.P.K
cosTheta2 = (x^2+y^2+l3^2-2*l3*(x*cos(phi)+y*sin(phi))-l1^2-l2^2)/(2*l1*l2);
if (-1 <= cosTheta2)&&(cosTheta2 <= 1)  % check if point is accessible
    % calculate sine of theta2 for the first solution using I.P.K
    sinTheta2_a = sqrt(1-cosTheta2^2);
    % calculate sine of theta2 for the second solution using I.P.K
    sinTheta2_b = -1*sinTheta2_a;
    theta2_a = atan2(sinTheta2_a, cosTheta2); % calculate theta 2 for the first solution
    theta2_b = atan2(sinTheta2_b, cosTheta2); % calculate theta 2 for the second solution
    % calculate theta 1 for the first solution
    theta1_a = atan2(y-l3*sin(phi),x-l3*cos(phi)) - atan2(l2*sinTheta2_a, l1+l2*cosTheta2);
    % calculate theta 1 for the second solution
    theta1_b = atan2(y-l3*sin(phi),x-l3*cos(phi)) - atan2(l2*sinTheta2_b, l1+l2*cosTheta2);
    % calculate theta 3 for the first solution
    theta3_a = phi - theta1_a - theta2_a;
    % calculate theta 3 for the second solution
    theta3_b = phi - theta1_b - theta2_b;
    % assign the solution 1 to the config matrix (1st column vector)
    config(:,1) = [theta1_a; theta2_a; theta3_a];
    % assign the solution 2 to the config matrix (2nd column vector)
    config(:,2) = [theta1_b; theta2_b; theta3_b];
```

```
else
    config = "point cannot be reached!";

end
end



% a function that computes the torque for random configruations generated
% in another function.
function torque = RRR_torque_2D(force,numberOfConfigs,config,Link_Lengths)
torque = zeros(3,numberOfConfigs); % declare the torque matrix
l1 = Link_Lengths(1); % assign link 1
l2 = Link_Lengths(2); % assign link 2
l3 = Link_Lengths(3); % assign link 3
for i = 1:numberOfConfigs
    % compute sine and cosine for the J_transport matrix
    s1 = sin(config(1,i));
    s12 = sin(config(1,i)+config(2,i));
    s123 = sin(config(1,i)+config(2,i)+config(3,i));
    c1 = cos(config(1,i));
    c12 = cos(config(1,i)+config(2,i));
    c123 = cos(config(1,i)+config(2,i)+config(3,i));
    J_transport = [-l1*s1-l2*s12-l3*s123,   l1*c1+l2*c12+l3*c123;
                   -l3*s123-l2*s12,        l3*c123+l2*c12;
                   -l3*s123,               l3*c123];
    %compute the torque using equation of tau = J_t*force
    torque(:,i) = J_transport*force;
end
end



% a function that takes #. config. and end eff. angle as the input, and output the configs.
function config_for_d = RRR_random_config(numberOfConfig,phi)
% angleConverter = 180/pi;  % convert radian to angles
% assign the angle limits.
lowerTheta1Limit = 0;
upperTheta1Limit = 3*pi/4;
lowerTheta2Limit = 3*pi/2;
upperTheta2Limit = 2*pi;
lowerTheta3Limit = 3*pi/2;
upperTheta3Limit = 2*pi;
config_for_d = zeros(3,numberOfConfig); % delcare a matrix for storing future random configs.
for i = 1:numberOfConfig
    theta3 = 0; % assign a dummy value to theta3 in order to get into the while loop
    while ~((lowerTheta3Limit<=theta3)&&(theta3<=upperTheta3Limit)) %check if theta3 is valid
        randNumber = rand(1,1); % assign a random number between 0 and 1.
        %generate the random angles using equation: lower+(upper-lower)*rand
        theta1 = lowerTheta1Limit+(upperTheta1Limit-lowerTheta1Limit)*randNumber;
        theta2 = lowerTheta2Limit+(upperTheta2Limit-lowerTheta2Limit)*randNumber;
        theta3 = (phi-theta1-(theta2-2*pi))+2*pi; % compute the theta3
    end
    config_for_d(:,i) = [theta1;theta2;theta3];% assign valid angles to matrix as the return
end
end
```

```
test for direct kinematics(phi in angles, not radians):
    0.0830    0.0784   78.2143

test for inverse kinematics, with 2 sets of solutions(elbow up and elbow down)
    0.5236    0.7575
    0.3927   -0.3927
    0.4488    1.0003

5 random configurations:
    1.7057    1.8208    1.2297    1.8881    1.4118
    5.8495    5.9262    5.5322    5.9711    5.6536
    5.1108    4.9190    5.9042    4.8068    5.6006

5 sets of torque corresponding to random configs:
   -0.0340   -0.0346   -0.0238   -0.0346   -0.0289
   -0.0139   -0.0148   -0.0056   -0.0150   -0.0095
   -0.0000    0.0000    0.0000   -0.0000    0.0000
```

Published with MATLAB® R2018b