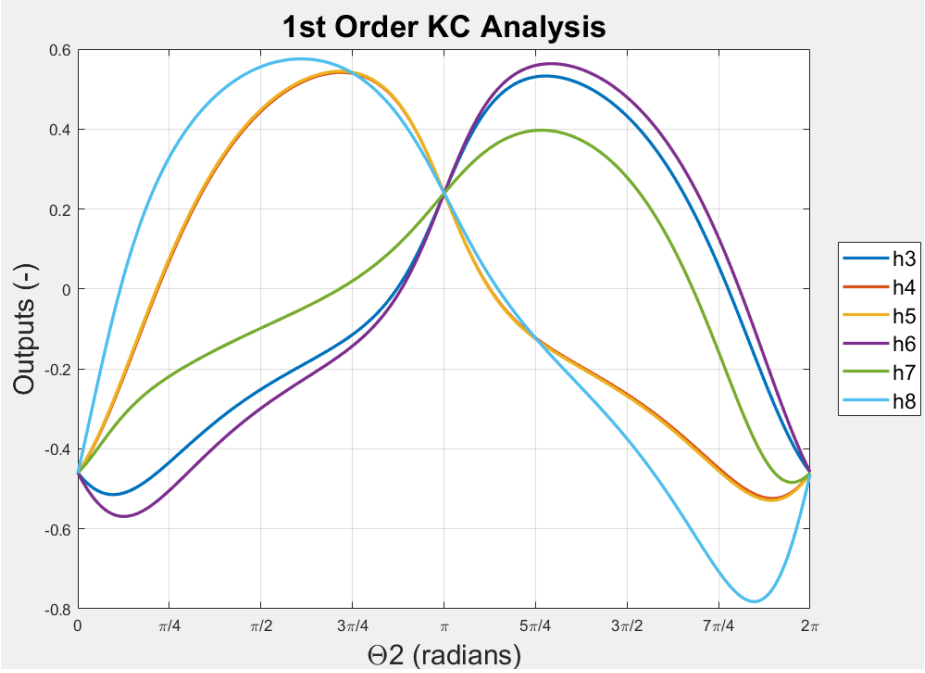
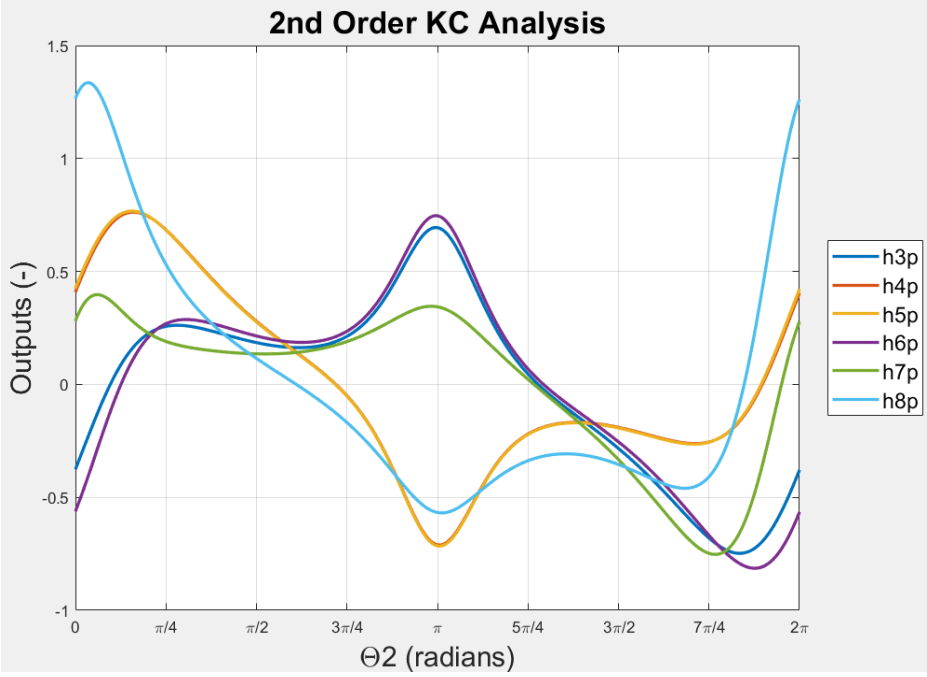
Semester Project Part III:

1st & 2nd Order Kinematic Coefficients (KC's)

ME 4133: Machine Design I, Fall 2022

Karan Baker, Jacob Coleman, Dylan Dupre, Donovan Gegg, William Hidalgo, Trieu Pham





Abstract

In this report, the team calculated and plotted the 1st and 2nd Order Kinematic Coefficients (KCs) for the ornithopter project before determining the Time Ratio (TR) for the mechanism, which was 1.063. To get the values for the 1st and 2nd Order KC plots, the Matlab “Solve” Function from Matlab’s Symbolic Toolbox was utilized in conjunction with the associated KC VLE. This was done with input angles of 0 to 360 degrees on 0.5 degree intervals. To validate the 1st Order KC’s curves, their X intercepts were compared with the maximum and minimum values of the corresponding position plots. To validate the 2nd Order KC’s curves, their X intercepts were compared with the maximum and minimum values of the corresponding 1st Order KC plots. The calculated error for all validation tests was 0.0044, which can be attributed to the estimation error leftover from when the X intercepts were found.

Table of Contents

Matlab 1st & 2nd Order Kinematic Coefficient (KC) Plots

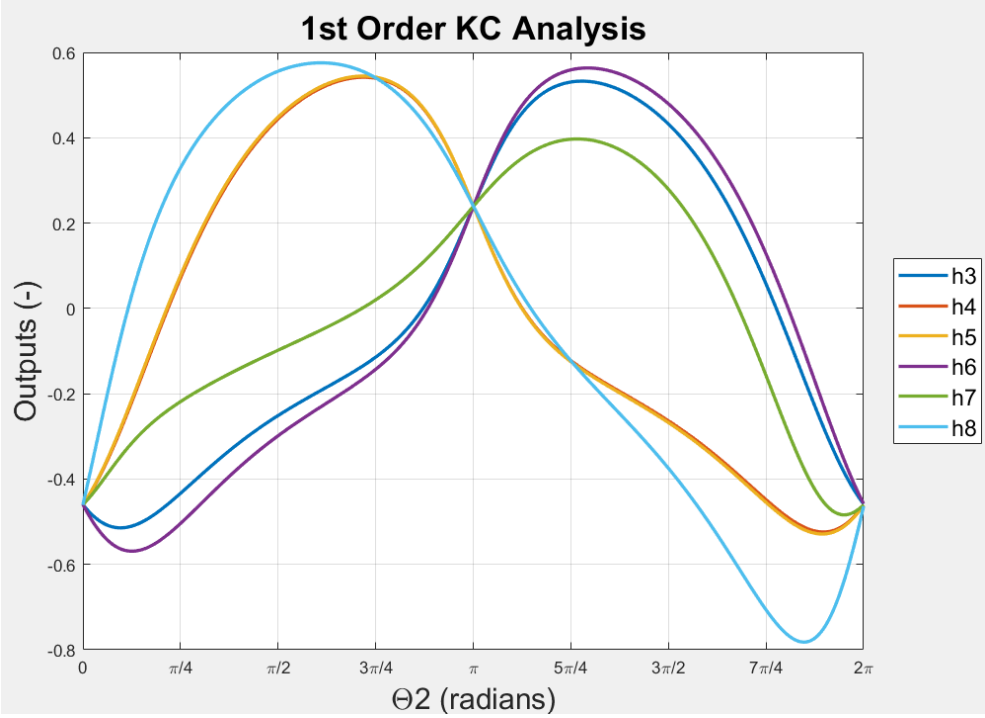


Figure 1: 1st Order KC Plots Generated with Matlab's "Solve" Function

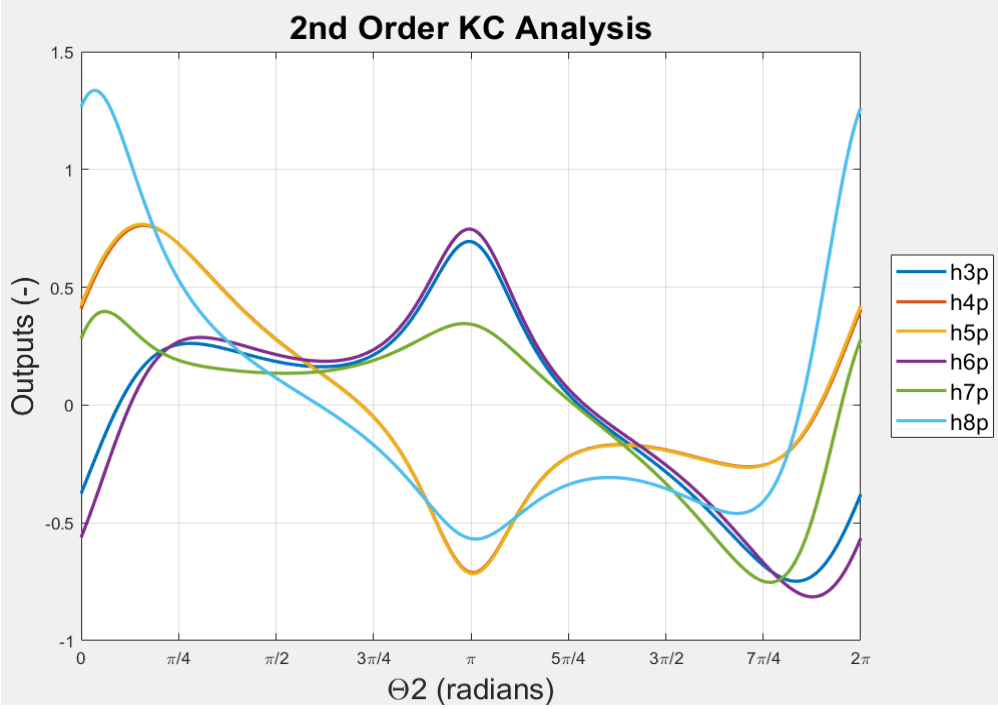


Figure 2: 2nd Order KC Plots Generated with Matlab's "Solve" Function

Matlab 1st and 2nd Order KC Analysis:

- The position solutions from *Semester Project Part II: Position Solutions* were utilized solving the 1st and 2nd Order KC's in this report.
- Matlab's *Symbolic Math* Toolbox was used to find solutions to the 1st and 2nd Order Vector Loop Equations (VLE's) by declaring the associated output h and h' variables with "syms" and using the "solve" function [1]. When plotted, these solutions yielded the 1st and 2nd Order KC curves in *Figure 1* and *Figure 2* respectively.
- Notably for the 1st Order KC's in *Figure 1*:
  - The h4 and h5 values at all input Θ's was essentially the same. The h3 and h6 values were also close between 7π/8 and 9π/8 (the absolute difference was ~0.0178 for both curves at these inputs).
  - H4, h5, and h8 had a maximum between 0 and π and a minimum between π and 2π. H3, h6, and h7 had the opposite behavior.
  - All 1st order KC's converged at π, with a value of ~0.2383.
- Notably for the 2nd Order KC's in *Figure 2*:
  - The h4' and h5' values at all input Θ's was essentially the same.
  - H4', h5', and h8' had 2 local minimums and 1 local maximum each. H3', h6', and h7' had 2 local maximums and 1 local minimum each.
  - All 2nd order KC's had either a maximum or minimum at π. H4', h5', and h8' had an absolute minimum at this input angle and h3' and h6' had an absolute maximum. H7' also had a local maximum.

Matlab 1st and 2nd Order KC Validation:

- To validate the 1st Order KC values, the absolute maximum and minimum X values for the given output position angle were compared to the X intercepts of its associated 1st Order KC curve. It was found that:
  - The difference between these values was directly dependent upon the resolution of the output angles solved. With input angles of 0 to 360 degrees on 0.5 degree intervals, the absolute difference of this value for all 1st order KC's was 0.0044.
- The same process was used to validate the 2nd Order KC values, with the absolute maximum and minimum X values for the given 1st Order KC being compared to the X intercepts of its associated 2nd Order KC curve.
  - With input angles of 0 to 360 degrees on 0.5 degree intervals, the absolute difference of this value for all 2nd order KC's was also 0.0044.
- Since the absolute difference in the validation process for both the 1st and 2nd Order KC's was the exact same, it can be concluded that this difference was due to the rounded errors associated with the function designed to approximate the X intercepts for the given curve.

Ornithopter Wing Time Ratio Calculation:

- The time ratio (TR) for a mechanism can be defined as the ratio of the change in input angle for the forward working stroke to its return stroke [2].
- Since the motion of the Ornithopter Wing tip is what is doing useful work, the difference between the input angle at the absolute maximum and minimum positions for link 8 of the mechanism were considered to find this value.
- The input angle values used to find the TR were as follows:

	Input angle Θ2 (rad)
At Maximum Θ8 Position (rad)	3.6023
At Minimum Θ8 Position (rad)	0.3663

Table 1: Input Angles for the Maximum and Minimum Link 8 Output Values

- The TR is thus:

$$TR = \frac{Forward\ Stroke\ \theta}{Return\ Stroke\ \theta} = \frac{\theta_{8max} - \theta_{8min}}{6.28 - (\theta_{8max} - \theta_{8min})} = \frac{3.2359\ rad}{3.0441\ rad} = 1.063$$

Equation 1: Time Ratio

References

[1] “EQN.” *Equations and Systems Solver - MATLAB*, https://www.mathworks.com/help/symbolic/sym.solve.html.

[2] *Quick-Return Mechanism Design and Analysis Projects*. https://journals.sagepub.com/doi/abs/10.7227/IJMEE.32.2.2.

Appendix

Appendix A: Matlab Code to Make Figures 1 and 2

```
%%%%% Donovan Gegg
% 11/2/2022
% Code for ME 4133 Machine Design I
% Project III: Orithopter Range of Motion
% Code Purpose: Determine other link angle positions given an input
% Angle. Plot these positions vs the input angle, theta 2. Plot the 1st and
% 2nd Order KC's for these angles once found. Validate all answers.

% Used Matlab's Symbolic Toolbox Solve function to determine potential angle
solutions
% for the given VLE's. 2 sets of solutions were given per Angle VLE. 1 set of
% Solutions were given per KC VLE. To get a continuous curve, the difference
% for the position graphs, the difference between current angle and previous angle
% for the given array had to be considered. If this difference was greater
% than 0.1, it was determined that that solution was in a different
% quadrant than the previous solution, and the other solution was used
% instead. The final value for each position array was removed because neither
% solution followed the given curve's trend. The 1st index for each position array
% was then set as the same as the final index for each array. For a final
% validation for the positions, all solutions were substituted in their associated
VLE to
% ensure that they were actually true, before being plotted.

% To validate the 1st order KC's, the maximums and minimums of the
% Position curves were compared to the estimated x intercept of the
% corresponding 1st order KC curve. The difference between all values was
% found to be 0.0532, which can be owed to rounding/zero estimation errors.
% Likewise, to validate the 2nd order KC's, the maximums and minimums of the
% 1st Order KC curves were compared to the estimated x intercept of the
% corresponding 2nd Order KC curve. The difference between all values was
% also found to be 0.0532.

% Clearing all previous values and figures
clc
clear
close all

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Initializating parameters.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Initializing csts/ knowns. R in mm. Angle in rad.
R1 = 59.7;
R2 = 18.8;
R3 = 41.0;
R4 = 40.1;
R_EC=12.7;
R_FC=137.2;
R5=136.4;
R6=11.9;
```

```
R7=15.6;
R8=11.7;
R_GI=13.1;
theta1 = 0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PROJECT PART II START
%%% The angles solved for Part II are NECESSARY for solving the KC's in
%%% Parts III and IV.
%%% Graders, skip this and scroll to the bottom. You've graded this part
%%% already.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Input angle array. An array to account for ALL possible input angles.
% Uncomment the one that will be used. More values means more accurate
% curves, but will take longer to solve.
%%% MUCH More Accurate, BUT Time consuming!
% theta2=linspace(0,360.5,722);
%%% Less Accurate (Default)
theta2=linspace(0,366,61);

% Converting theta values to radians, which capture quadrant positions.
for i = 1:length(theta2)
    theta2(i) = theta2(i)*3.14/180;
end

% Output angle arrays. Arrays to account for ALL possible output angles
% for the associated input angle. Each has same dimensions as input array.
% Values will be put in the associated slot as it is solved for.
% Required for plotting.
theta3_arr = linspace(0,0,length(theta2));
theta4_arr = linspace(0,0,length(theta2));
theta5_arr = linspace(0,0,length(theta2));
theta6_arr = linspace(0,0,length(theta2));
theta7_arr = linspace(0,0,length(theta2));
theta8_arr = linspace(0,0,length(theta2));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SOLVING for output angles
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Looping through all possible input angles
disp('////////////////////////////////')
disp('// Solving for output angles...')
disp('////////////////////////////////')

for i = 1:length(theta2)
    msg = sprintf('Solving Positions for Theta2 = %d radians.',theta2(i));
    disp(msg)
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%% Solving for theta3 and theta4 (rad)
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Setting variables to solve for...
    syms theta3 theta4
    % VLEs
    E1 = R2*cos(theta2(i))+R3*cos(theta3)-R4*cos(theta4)-R1*cos(theta1) == 0;
    E2 = R2*sin(theta2(i))+R3*sin(theta3)-R4*sin(theta4)-R1*sin(theta1) == 0;
```

```
% Solving VLEs...
[theta3,theta4] = solve(E1,E2,theta3,theta4);
% Adjusting for potential angle wrap
if i > 2
    if abs(theta3_arr(i-1)-vpa(theta3(1))) > 0.1
        theta3 = vpa(theta3(2));
        theta4 = vpa(theta4(2));
    else
        theta3 = vpa(theta3(1));
        theta4 = vpa(theta4(1));
    end
else
    theta3 = vpa(theta3(1));
    theta4 = vpa(theta4(1));
end
% Updating the output arrays...
theta3_arr(i) = theta3;
theta4_arr(i) = theta4;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Solving for theta5 and theta6 (rad)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Some angles have correlations with other angles.
theta_EC_off = 19.275*3.14/180;
theta_FC_off = 9.736*3.14/180;
theta_EC = theta3_arr(i) + theta_EC_off; % cst
theta_FC = theta4_arr(i) + theta_FC_off; % cst
% Setting variables to solve for...
syms theta5 theta6
% VLEs
E1 = R_EC*cos(theta_EC)+R_FC*cos(theta_FC)-R6*cos(theta6)-R5*cos(theta5) == 0;
E2 = R_EC*sin(theta_EC)+R_FC*sin(theta_FC)-R6*sin(theta6)-R5*sin(theta5) == 0;
% Solving VLEs...
[theta5,theta6] = solve(E1,E2,theta5,theta6);
% Adjusting for potential angle wrap
if i > 2
    if abs(theta5_arr(i-1)-vpa(theta5(1))) > 0.1
        theta5 = vpa(theta5(2));
        theta6 = vpa(theta6(2));
    else
        theta5 = vpa(theta5(1));
        theta6 = vpa(theta6(1));
    end
else
    theta5 = vpa(theta5(1));
    theta6 = vpa(theta6(1));
end
% Updating the output arrays...
theta5_arr(i) = theta5;
theta6_arr(i) = theta6;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Solving for theta7 and theta8 (rad)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Some angles have correlations with other angles.
theta_GI = theta5_arr(i); % Literally the same link.
```

```
    % Setting variables to solve for...
    syms theta7 theta8
    % VLEs
    E1 = R7*cos(theta7)-R8*cos(theta8)-R_GI*cos(theta_GI)+R6*cos(theta6) == 0;
    E2 = R7*sin(theta7)-R8*sin(theta8)-R_GI*sin(theta_GI)+R6*sin(theta6) == 0;
    % Solving VLEs...
    [theta7,theta8] = solve(E1,E2,theta7,theta8);
    % Adjusting for potential angle wrap
    if i > 2
        if abs(theta7_arr(i-1)-vpa(theta7(1))) > 0.1
            theta7 = vpa(theta7(2));
            theta8 = vpa(theta8(2));
        else
            theta7 = vpa(theta7(1));
            theta8 = vpa(theta8(1));
        end
    else
        theta7 = vpa(theta7(1));
        theta8 = vpa(theta8(1));
    end
    % Updating the output arrays...
    theta7_arr(i) = theta7;
    theta8_arr(i) = theta8;
end
% Removing the last index from arrays b/c solutions
% don't follow the curve solutions at same theta2 +/- 2pi position.
% Replacing 1st index of each array with the 2pi index
% Theta 2.
theta2(length(theta2))=[];
theta2(1)=0;
% Theta 3
theta3_arr(length(theta3_arr))=[];
theta3_arr(1)=theta3_arr(length(theta3_arr));
% Theta 4
theta4_arr(length(theta4_arr))=[];
theta4_arr(1)=theta4_arr(length(theta4_arr));
% Theta 5
theta5_arr(length(theta5_arr))=[];
theta5_arr(1)=theta5_arr(length(theta5_arr));
% Theta 6
theta6_arr(length(theta6_arr))=[];
theta6_arr(1)=theta6_arr(length(theta6_arr));
% Theta 7
theta7_arr(length(theta7_arr))=[];
theta7_arr(1)=theta7_arr(length(theta7_arr));
% Theta 8
theta8_arr(length(theta8_arr))=[];
theta8_arr(1)=theta8_arr(length(theta8_arr));
%%%%%%%%%%%%%%%%%%
% Data Validation
%%%%%%%%%%%%%%%%%%
% Need to check each value in the arrays to ensure
% that they satisfy the VLEs.

disp('////////////////////////////////')
```

```
disp('// Validating output angles...')
disp('////////////////////////////////')

% Initializing tolerance and failure counter
check_fails = 0;
tol = 0.00005; % Above or below zero for each VLE
% Looping through entire array
for i = 2:length(theta2)
    % VLE 1 Check
    E1 = R2*cos(theta2(i))+R3*cos(theta3_arr(i))-R4*cos(theta4_arr(i))-
R1*cos(theta1);
    E2 = R2*sin(theta2(i))+R3*sin(theta3_arr(i))-R4*sin(theta4_arr(i))-
R1*sin(theta1);
    if abs(E1) > tol || abs(E2) > tol
        check_fails = check_fails + 1;
    end
    % VLE 2 Check
    theta_EC = theta3_arr(i) + theta_EC_off; % cst
    theta_FC = theta4_arr(i) + theta_FC_off; % cst
    E1 = R_EC*cos(theta_EC)+R_FC*cos(theta_FC)-R6*cos(theta6_arr(i))-
R5*cos(theta5_arr(i));
    E2 = R_EC*sin(theta_EC)+R_FC*sin(theta_FC)-R6*sin(theta6_arr(i))-
R5*sin(theta5_arr(i));
    if abs(E1) > tol || abs(E2) > tol
        check_fails = check_fails + 1;
    end
    % VLE 3 Check
    theta_GI = theta5_arr(i); % Literally the same link.
    E1 = R7*cos(theta7_arr(i))-R8*cos(theta8_arr(i))-
R_GI*cos(theta_GI)+R6*cos(theta6_arr(i));
    E2 = R7*sin(theta7_arr(i))-R8*sin(theta8_arr(i))-
R_GI*sin(theta_GI)+R6*sin(theta6_arr(i));
    if abs(E1) > tol || abs(E2) > tol
        check_fails = check_fails + 1;
    end
end
% Displaying validation results in cmd window
if check_fails == 0
    disp("ALL angles validated!")
else
    disp("Some angles failed!")
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% PROJECT PART II END
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PROJECT PART III START
%%% 1st Order KC's
%%% Mostly repurposed Part II code, hence why they look
%%% basically the same.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Output 1st Order KC arrays. Arrays to account for ALL possible 1st Order
% KC's for the associated input angle. Each has same dimensions as input array.
```

```
% Values will be put in the associated slot as it is solved for.
% Required for plotting.
h3_arr = linspace(0,0,length(theta2));
h4_arr = linspace(0,0,length(theta2));
h5_arr = linspace(0,0,length(theta2));
h6_arr = linspace(0,0,length(theta2));
h7_arr = linspace(0,0,length(theta2));
h8_arr = linspace(0,0,length(theta2));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SOLVING for output 1st Order KC's
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp('////////////////////////////////')
disp("// Solving for 1st Order KC's...")
disp('////////////////////////////////')

% Looping through all possible input angles
for i = 1:length(theta2)
    msg = sprintf('Solving 1st Order KCs for Theta2 = %d radians.',theta2(i));
    disp(msg)
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%% Solving for h3 and h4
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Setting variables to solve for...
    syms h3 h4
    % VLEs
    E1 = -R2*sin(theta2(i))-R3*sin(theta3_arr(i))*h3+R4*sin(theta4_arr(i))*h4 == 0;
    E2 = R2*cos(theta2(i))+R3*cos(theta3_arr(i))*h3-R4*cos(theta4_arr(i))*h4 == 0;
    % Solving VLEs...
    [h3,h4] = solve(E1,E2,h3,h4);

    % Updating the output arrays...
    h3_arr(i) = h3;
    h4_arr(i) = h4;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%% Solving for h5 and h6
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Some angles have correlations with other angles.
    theta_EC_off = 19.275*3.14/180;
    theta_FC_off = 9.736*3.14/180;
    theta_EC = theta3_arr(i) + theta_EC_off; % cst
    theta_FC = theta4_arr(i) + theta_FC_off; % cst
    % Setting variables to solve for...
    syms h5 h6
    % VLEs
    E1 = -R_EC*sin(theta_EC)*h3_arr(i)-
R_FC*sin(theta_FC)*h4_arr(i)+R6*sin(theta6_arr(i))*h6+R5*sin(theta5_arr(i))*h5 == 0;
    E2 = R_EC*cos(theta_EC)*h3_arr(i)+R_FC*cos(theta_FC)*h4_arr(i)-
R6*cos(theta6_arr(i))*h6-R5*cos(theta5_arr(i))*h5 == 0;
    % Solving VLEs...
    [h5,h6] = solve(E1,E2,h5,h6);

    % Updating the output arrays...
    h5_arr(i) = h5;
    h6_arr(i) = h6;
```

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%% Solving for h7 and h8
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Some angles have correlations with other angles.
    theta_GI = theta5_arr(i); % Literally the same link.

    % Setting variables to solve for...
    syms h7 h8
    % VLEs
    E1 = -
R7*sin(theta7_arr(i))*h7+R8*sin(theta8_arr(i))*h8+R_GI*sin(theta_GI)*h5_arr(i)-
R6*sin(theta6_arr(i))*h6_arr(i) == 0;
    E2 = R7*cos(theta7_arr(i))*h7-R8*cos(theta8_arr(i))*h8-
R_GI*cos(theta_GI)*h5_arr(i)+R6*cos(theta6_arr(i))*h6_arr(i) == 0;
    % Solving VLEs...
    [h7,h8] = solve(E1,E2,h7,h8);

    % Updating the output arrays...
    h7_arr(i) = h7;
    h8_arr(i) = h8;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% 2nd Order KC's
%%% Literally the same as 1st order, but different VLEs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Output 2nd Order KC arrays. Arrays to account for ALL possible 2nd Order
% KC's for the associated input angle. Each has same dimensions as input array.
% Values will be put in the associated slot as it is solved for.
% Required for plotting.
h3p_arr = linspace(0,0,length(theta2));
h4p_arr = linspace(0,0,length(theta2));
h5p_arr = linspace(0,0,length(theta2));
h6p_arr = linspace(0,0,length(theta2));
h7p_arr = linspace(0,0,length(theta2));
h8p_arr = linspace(0,0,length(theta2));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SOLVING for output 2nd Order KC's
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp('////////////////////////////////')
disp("// Solving for 2nd Order KC's...")
disp('////////////////////////////////')

% Looping through all possible input angles
for i = 1:length(theta2)
    msg = sprintf('Solving 2nd Order KCs for Theta2 = %d radians.',theta2(i));
    disp(msg)
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%% Solving for h3p and h4p
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Setting variables to solve for...
    syms h3p h4p
    % VLEs
```

```
    E1 = -R2*cos(theta2(i))-R3*cos(theta3_arr(i))*(h3_arr(i)^2)-
R3*sin(theta3_arr(i))*h3p+R4*cos(theta4_arr(i))*(h4_arr(i)^2)+R4*sin(theta4_arr(i))*h
4p == 0;
    E2 = -R2*sin(theta2(i))-
R3*sin(theta3_arr(i))*(h3_arr(i)^2)+R3*cos(theta3_arr(i))*h3p+R4*sin(theta4_arr(i))*(
h4_arr(i)^2)-R4*cos(theta4_arr(i))*h4p == 0;

    % Solving VLEs...
    [h3p,h4p] = solve(E1,E2,h3p,h4p);

    % Updating the output arrays...
    h3p_arr(i) = h3p;
    h4p_arr(i) = h4p;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%% Solving for h5p and h6p
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Some angles have correlations with other angles.
    theta_EC_off = 19.275*3.14/180;
    theta_FC_off = 9.736*3.14/180;
    theta_EC = theta3_arr(i) + theta_EC_off; % cst
    theta_FC = theta4_arr(i) + theta_FC_off; % cst
    % Setting variables to solve for...
    syms h5p h6p
    % VLEs
    E1 = -R_EC*cos(theta_EC)*(h3_arr(i)^2)-R_EC*sin(theta_EC)*h3p_arr(i)-
R_FC*cos(theta_FC)*(h4_arr(i)^2)-
R_FC*sin(theta_FC)*h4p_arr(i)+R6*cos(theta6_arr(i))*(h6_arr(i)^2)+R6*sin(theta6_arr(i
))*h6p+R5*cos(theta5_arr(i))*(h5_arr(i)^2)+R5*sin(theta5_arr(i))*h5p  == 0;
    E2 = -R_EC*sin(theta_EC)*(h3_arr(i)^2)+R_EC*cos(theta_EC)*h3p_arr(i)-
R_FC*sin(theta_FC)*(h4_arr(i)^2)+R_FC*cos(theta_FC)*h4p_arr(i)+R6*sin(theta6_arr(i))*
(h6_arr(i)^2)-R6*cos(theta6_arr(i))*h6p+R5*sin(theta5_arr(i))*(h5_arr(i)^2)-
R5*cos(theta5_arr(i))*h5p == 0;

    % Solving VLEs...
    [h5p,h6p] = solve(E1,E2,h5p,h6p);

    % Updating the output arrays...
    h5p_arr(i) = h5p;
    h6p_arr(i) = h6p;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%% Solving for h7p and h8p
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Some angles have correlations with other angles.
    theta_GI = theta5_arr(i); % Literally the same link.

    % Setting variables to solve for...
    syms h7p h8p
    % VLEs
    E1 = -R7*cos(theta7_arr(i))*(h7_arr(i)^2)-
R7*sin(theta7_arr(i))*h7p+R8*cos(theta8_arr(i))*(h8_arr(i)^2)+R8*sin(theta8_arr(i))*h
8p+R_GI*cos(theta5_arr(i))*(h5_arr(i)^2)+R_GI*sin(theta5_arr(i))*h5p_arr(i)-
R6*cos(theta6_arr(i))*(h6_arr(i)^2)-R6*sin(theta6_arr(i))*h6p_arr(i) == 0;
    E2 = -
R7*sin(theta7_arr(i))*(h7_arr(i)^2)+R7*cos(theta7_arr(i))*h7p+R8*sin(theta8_arr(i))*(
h8_arr(i)^2)-R8*cos(theta8_arr(i))*h8p+R_GI*sin(theta5_arr(i))*(h5_arr(i)^2)-
```

```
R_GI*cos(theta5_arr(i))*h5p_arr(i)-
R6*sin(theta6_arr(i))*(h6_arr(i)^2)+R6*cos(theta6_arr(i))*h6p_arr(i) == 0;

    % Solving VLEs...
    [h7p,h8p] = solve(E1,E2,h7p,h8p);

    % Updating the output arrays...
    h7p_arr(i) = h7p;
    h8p_arr(i) = h8p;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% 1st and 2nd Order KC Data Validation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp('////////////////////////////////')
disp("// Validating 1st Order KC's...")
disp('////////////////////////////////')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 1st Order KC's Validations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% IF VALID, the x values for the max or min for the
% Position curves must be sufficiently close to the
% x ints for the corresponding 1st Order KC curve.

%%% Position
% Finding X Position Minimums indices
local_min_theta3 = islocalmin(theta3_arr);
local_min_theta4 = islocalmin(theta4_arr);
local_min_theta5 = islocalmin(theta5_arr);
local_min_theta6 = islocalmin(theta6_arr);
local_min_theta7 = islocalmin(theta7_arr);
local_min_theta8 = islocalmin(theta8_arr);


% Finding X Position Maximums indices
local_max_theta3 = islocalmax(theta3_arr);
local_max_theta4 = islocalmax(theta4_arr);
local_max_theta5 = islocalmax(theta5_arr);
local_max_theta6 = islocalmax(theta6_arr);
local_max_theta7 = islocalmax(theta7_arr);
local_max_theta8 = islocalmax(theta8_arr);

% X Position Min/Max lists. 1 Max and Min for each.
max_min_theta3 = [theta2(local_min_theta3),theta2(local_max_theta3)];
max_min_theta4 = [theta2(local_min_theta4),theta2(local_max_theta4)];
max_min_theta5 = [theta2(local_min_theta5),theta2(local_max_theta5)];
max_min_theta6 = [theta2(local_min_theta6),theta2(local_max_theta6)];
max_min_theta7 = [theta2(local_min_theta7),theta2(local_max_theta7)];
max_min_theta8 = [theta2(local_min_theta8),theta2(local_max_theta8)];

%%% 1st Order KC's
h3_zeros = x_Int_Estimator(theta2,h3_arr);
h4_zeros = x_Int_Estimator(theta2,h4_arr);
```

```
h5_zeros = x_Int_Estimator(theta2,h5_arr);
h6_zeros = x_Int_Estimator(theta2,h6_arr);
h7_zeros = x_Int_Estimator(theta2,h7_arr);
h8_zeros = x_Int_Estimator(theta2,h8_arr);

% Calculating the diffs between each Min-Max's and the x-ints
value_diffs1 = linspace(0,0,12);
value_diffs1(1) = abs(max_min_theta3(1)-h3_zeros(1));
value_diffs1(2) = abs(max_min_theta3(2)-h3_zeros(2));
value_diffs1(3) = abs(max_min_theta4(1)-h4_zeros(1));
value_diffs1(4) = abs(max_min_theta4(2)-h4_zeros(2));
value_diffs1(5) = abs(max_min_theta5(1)-h5_zeros(1));
value_diffs1(6) = abs(max_min_theta5(2)-h5_zeros(2));
value_diffs1(7) = abs(max_min_theta6(1)-h6_zeros(1));
value_diffs1(8) = abs(max_min_theta6(2)-h6_zeros(2));
value_diffs1(9) = abs(max_min_theta7(1)-h7_zeros(1));
value_diffs1(10) = abs(max_min_theta7(2)-h7_zeros(2));
value_diffs1(11) = abs(max_min_theta8(1)-h8_zeros(1));
value_diffs1(12) = abs(max_min_theta8(2)-h8_zeros(2));

% Checking if angle diffs is sufficiently large
check_fails = 0;
for i = 1:length(value_diffs1)
    if value_diffs1(i) > 0.1
        check_fails = check_fails + 1;
        disp(value_diffs1(i))
    end
end

% Displaying result
if check_fails > 0
    disp("Some 1st Order KC's are wrong!")
    disp(check_fails)
else
    disp("All 1st Order KC's Validated!")
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 2nd Order KC's Validations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% IF VALID, the x values for the max or min for the
% 1st Order Kc's curves must be sufficiently close to the
% x ints for the corresponding 2nd Order KC curve.

disp('//////////////////////////////////')
disp("// Validating 2nd Order KC's...")
disp('//////////////////////////////////')

%%% Position
% Finding X 1st Order KC Minimums indices
local_min_h3 = islocalmin(h3_arr);
local_min_h4 = islocalmin(h4_arr);
local_min_h5 = islocalmin(h5_arr);
local_min_h6 = islocalmin(h6_arr);
local_min_h7 = islocalmin(h7_arr);
```

```
local_min_h8 = islocalmin(h8_arr);

% Finding X 1st Order KC Maximums indices
local_max_h3 = islocalmax(h3_arr);
local_max_h4 = islocalmax(h4_arr);
local_max_h5 = islocalmax(h5_arr);
local_max_h6 = islocalmax(h6_arr);
local_max_h7 = islocalmax(h7_arr);
local_max_h8 = islocalmax(h8_arr);

% X Position Min/Max lists. 1 Max and Min for each.
max_min_h3 = [theta2(local_min_h3),theta2(local_max_h3)];
max_min_h4 = [theta2(local_min_h4),theta2(local_max_h4)];
max_min_h5 = [theta2(local_min_h5),theta2(local_max_h5)];
max_min_h6 = [theta2(local_min_h6),theta2(local_max_h6)];
max_min_h7 = [theta2(local_min_h7),theta2(local_max_h7)];
max_min_h8 = [theta2(local_min_h8),theta2(local_max_h8)];

%%% 2nd Order KC's
h3p_zeros = x_Int_Estimator(theta2,h3p_arr);
h4p_zeros = x_Int_Estimator(theta2,h4p_arr);
h5p_zeros = x_Int_Estimator(theta2,h5p_arr);
h6p_zeros = x_Int_Estimator(theta2,h6p_arr);
h7p_zeros = x_Int_Estimator(theta2,h7p_arr);
h8p_zeros = x_Int_Estimator(theta2,h8p_arr);

% Calculating the diffs between each Min-Max's and the x-ints
value_diffs2 = linspace(0,0,12);
value_diffs2(1) = abs(max_min_h3(1)-h3p_zeros(1));
value_diffs2(2) = abs(max_min_h3(2)-h3p_zeros(2));
value_diffs2(3) = abs(max_min_h4(1)-h4p_zeros(2));
value_diffs2(4) = abs(max_min_h4(2)-h4p_zeros(1));
value_diffs2(5) = abs(max_min_h5(1)-h5p_zeros(2));
value_diffs2(6) = abs(max_min_h5(2)-h5p_zeros(1));
value_diffs2(7) = abs(max_min_h6(1)-h6p_zeros(1));
value_diffs2(8) = abs(max_min_h6(2)-h6p_zeros(2));
value_diffs2(9) = abs(max_min_h7(1)-h7p_zeros(2));
value_diffs2(10) = abs(max_min_h7(2)-h7p_zeros(1));
value_diffs2(11) = abs(max_min_h8(1)-h8p_zeros(2));
value_diffs2(12) = abs(max_min_h8(2)-h8p_zeros(1));

% Checking if angle diffs is sufficiently large
check_fails = 0;
for i = 1:length(value_diffs2)
    if value_diffs2(i) > 0.1
        check_fails = check_fails + 1;
        disp(value_diffs2(i))
    end
end

% Displaying result
if check_fails > 0
    disp("Some 2nd Order KC's are wrong!")
    disp(check_fails)
else
```

```
    disp("All 2nd Order KC's Validated!")
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% PROJECT PART III END
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Plotting Values (Parts II and III)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Project Part II Graph
% Position Graph
% Plotting all the output angles vs theta2
figure('Name','Position Analysis')

% Tile layout for better legend
t1 = tiledlayout('flow','TileSpacing','compact');

% Plotting all the output angles vs theta2
nexttile
plot(theta2,theta3_arr, 'LineWidth', 2)
hold on
plot(theta2,theta4_arr, 'LineWidth', 2)
hold on
plot(theta2,theta5_arr, 'LineWidth', 2)
hold on
plot(theta2,theta6_arr, 'LineWidth', 2)
hold on
plot(theta2,theta7_arr, 'LineWidth', 2)
hold on
plot(theta2,theta8_arr, 'LineWidth', 2)
grid on

% Setting x boundaries.
pi = 3.14;
xlim([0,(2*pi)]);

% Creating x ticks and labels
xticks([0,pi/4,pi/2,3*pi/4,pi,5*pi/4,3*pi/2,7*pi/4,2*pi]);
xticklabels({'0','\pi/4','\pi/2','3\pi/4','\pi','5\pi/4','3\pi/2','7\pi/4','2\pi'});

% Labeling title, x axis, & y axis.
title('Position Analysis','FontSize',20);
xlabel('\Theta2 (radians)','FontSize',18);
ylabel('Outputs (radians)','FontSize',18);

% Adding legends
legend1 = legend('theta3','theta4','theta5','theta6','theta7','theta8');
legend1.Layout.Tile = "east";
legend1.FontSize = 14;

% Modifying figure window size to automatically fit legend.
set(gcf, 'Position', [100,100,850,600])
```

```
%%% Project Part III Graphs
%%% KC1 Graph
figure('Name','1st Order KC Analysis')

% Tile layout for better legend
t2 = tiledlayout('flow','TileSpacing','compact');

% Plotting all the 1st Order KC's vs theta2
nexttile
plot(theta2,h3_arr,'LineWidth', 2)
hold on
plot(theta2,h4_arr,'LineWidth', 2)
hold on
plot(theta2,h5_arr,'LineWidth', 2)
hold on
plot(theta2,h6_arr,'LineWidth', 2)
hold on
plot(theta2,h7_arr,'LineWidth', 2)
hold on
plot(theta2,h8_arr,'LineWidth', 2)
grid on

% Setting x boundaries.
pi = 3.14;
xlim([0,(2*pi)]);

% Creating x ticks and labels
xticks([0,pi/4,pi/2,3*pi/4,pi,5*pi/4,3*pi/2,7*pi/4,2*pi]);
xticklabels({'0','\pi/4','\pi/2','3\pi/4','\pi','5\pi/4','3\pi/2','7\pi/4','2\pi'});

% Labeling title, x axis, & y axis.
title('1st Order KC Analysis','FontSize',20);
xlabel('\Theta2 (radians)','FontSize',18);
ylabel('Outputs (-)','FontSize',18);

% Adding legends
legend2 = legend('h3','h4','h5','h6','h7','h8');
legend2.Layout.Tile = "east";
legend2.FontSize = 14;

% Modifying figure window size to automatically fit legend.
set(gcf, 'Position', [100,100,850,600])

%%% KC2 Graph
figure('Name','2nd Order KC Analysis')

% Tile layout for better legend
t3 = tiledlayout('flow','TileSpacing','compact');

% Plotting all the 1st Order KC's vs theta2
nexttile
plot(theta2,h3p_arr,'LineWidth', 2)
hold on
plot(theta2,h4p_arr,'LineWidth', 2)
hold on
```

```
plot(theta2,h5p_arr,'LineWidth', 2)
hold on
plot(theta2,h6p_arr,'LineWidth', 2)
hold on
plot(theta2,h7p_arr,'LineWidth', 2)
hold on
plot(theta2,h8p_arr,'LineWidth', 2)
grid on

% Setting x boundaries.
pi = 3.14;
xlim([0,(2*pi)]);

% Creating x ticks and labels
xticks([0,pi/4,pi/2,3*pi/4,pi,5*pi/4,3*pi/2,7*pi/4,2*pi]);
xticklabels({'0','\pi/4','\pi/2','3\pi/4','\pi','5\pi/4','3\pi/2','7\pi/4','2\pi'});

% Labeling title, x axis, & y axis.
title('2nd Order KC Analysis','FontSize',20);
xlabel('\Theta2 (radians)','FontSize',18);
ylabel('Outputs (-)','FontSize',18);

% Adding legends
legend3 = legend('h3p','h4p','h5p','h6p','h7p','h8p');
legend3.Layout.Tile = "east";
legend3.FontSize = 14;

% Modifying figure window size to automatically fit legend.
set(gcf, 'Position', [100,100,850,600])

%% Time Ratio Calculation
% The accuracy for this is dependent upon the accuracy of the zeros for
% h8 (the output), which is directly dependent upon the number
% of input angles.

% TR = change in input angle disp forward stroke /change in input angle
% disp backward stroke
TR_Top = max_min_theta8(2) - max_min_theta8(1);
TR = TR_Top/(6.28 - TR_Top);

%% Helper Function
% Function to find x intercepts and plot them.
function [x_zero] = x_Int_Estimator(x,y)
    % initializing counter/indexer and current values and empty vectors
    i = 1;
    j = 1;
    current_val = 0;
    zeros = [0,0];

    % Looping thru length of input vector
    while i <= length(y)
        % Recording previous value
        previous_val = current_val;
        % Setting current value
        current_val = y(i);
```

```
        % Comparing current and previous values
        if previous_val < 0 && current_val > 0
            % Averaging the points to estimate the x value
            est_zero = (x(i-1)+x(i))/2;
            % Concatenating vectors
            zeros(j) = est_zero;
            j = j+1;
        % Comparing current and previous values
        elseif previous_val > 0 && current_val < 0
            % Averaging the points to estimate the x value
            est_zero = (x(i-1)+x(i))/2;
            % Concatenating vectors
            zeros(j) = est_zero;
            j = j+1;
        end
        i = i + 1;
    end
    x_zero = zeros;
end
```