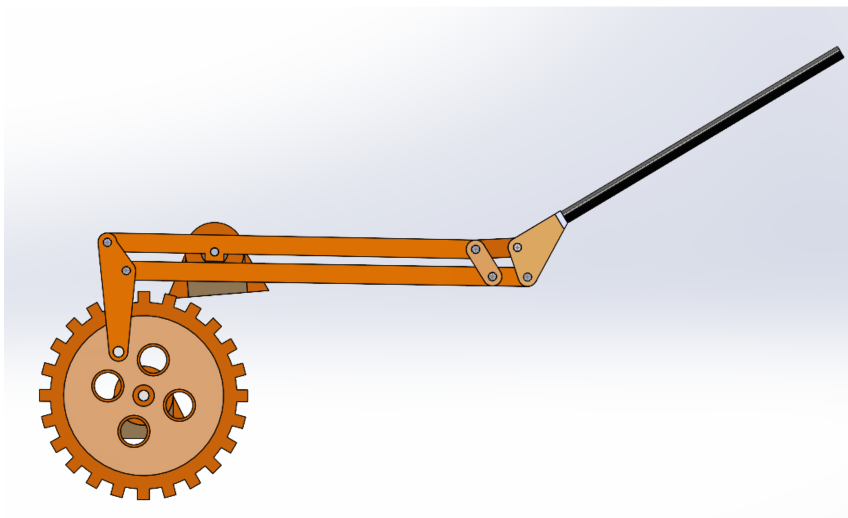
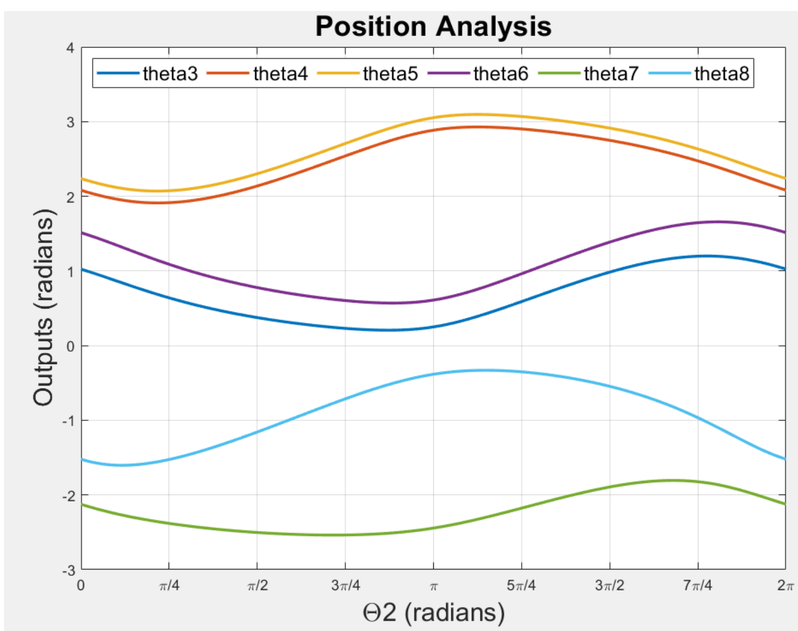
Semester Project Part II

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 determined and plotted the position solutions for this semester's project using the Newton Rhapson method as well as the “Solve” function in MATLAB. Given any input angle $\theta 2$ ranging from zero to 360 degrees, the angle solutions for all other links can now be determined. Additionally, the project mechanism was modeled, assembled, and animated in Solidworks as a form of validation. Excel was also used to help validate the results from the Solidworks assembly models.

<u>Table of Contents:</u>

Solidworks Ornithopter Generated Model & Analysis

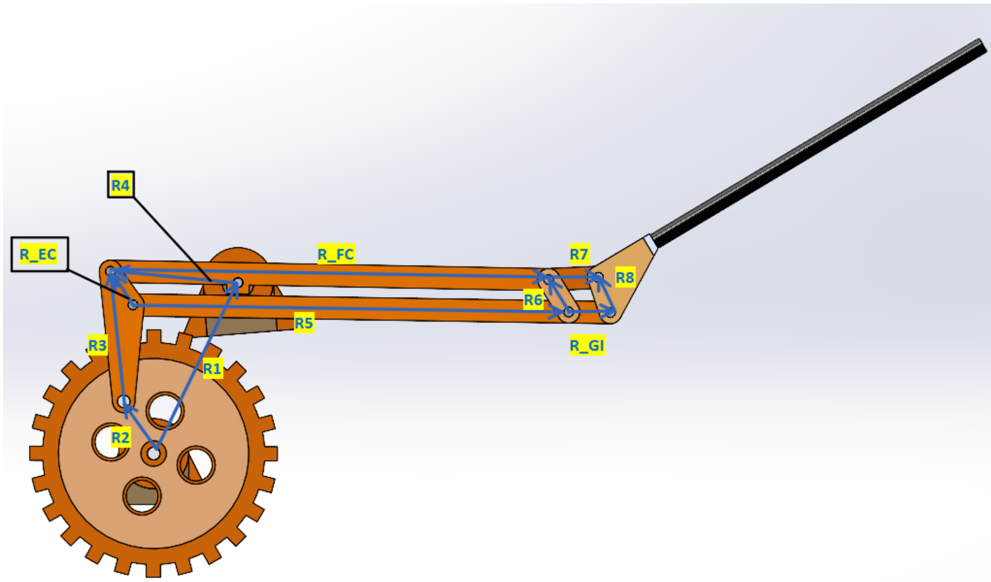


Figure 1: Ornithopter Model Generated in Solidworks Utilizing Provided Vector Lengths.

Table 1: Dimensions Utilized in SolidWorks Models			
R1 = 59.7 mm	R4 = 40.1 mm	R5 = 136.4 mm	R8 = 11.7 mm
R2 = 18.8 mm	R_EC = 12.7 mm	R6 = 11.9 mm	R_GI = 13.1 mm
R3 = 40.1 mm	R_FC = 136.4 mm	R7 = 15.6 mm	

Solidworks Analysis

- All team members generated ornithopter wings similar to *Figure 1* using the vector values seen in *Table 1*.
- It was found that the *distance* mate in Solidworks could be used in place of the physical gear seen in *Figure 1* to restrict the input link's path of motion.
- The team found that links R7 and R_FC were not the same physical link. When these were treated as a single link, the structure became rigid. When R7 and R_FC were treated as separate links, the team's ornithopter wings were then able to execute a continuous motion as the input link moved along the path of an imaginary gear which could be rotated 360 degrees.

Matlab Position Plots

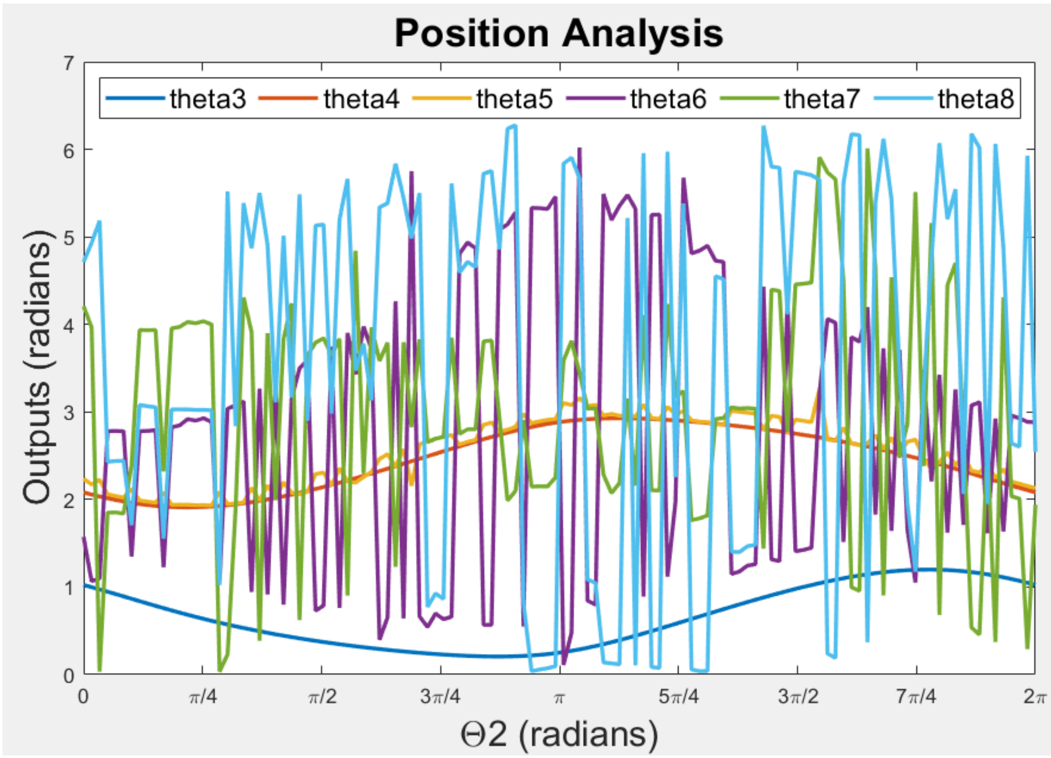


Figure 2: Plots Generated with Hard-Coded Newton Raphson Method.

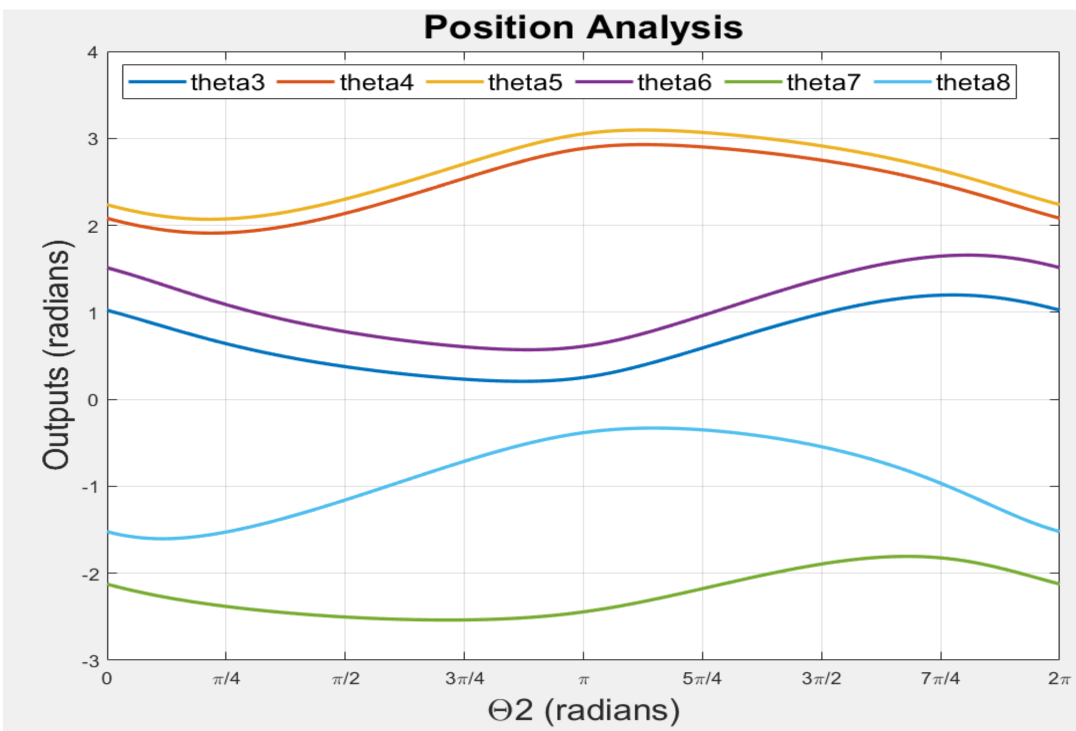


Figure 3: Plots Generated with Matlab's "Solve" Function

Matlab Position Plot Analysis:

- The team found that Newton Rhapson's method utilizing Cramer's Rule worked to solve Θ's 3 & 4, but had difficulty generating smooth outputs for Θ's 5 - 8, as seen in *Figure 2*. This happened even when the initial Θ's were changed when solving for these angles.
- Matlab's *Symbolic Math* Toolbox was used instead to find solutions to the Vector Loop Equations (VLE's) by declaring the output Θ variables with "syms" and using the "solve" function [1]. When plotted, these solutions yielded the curves in *Figure 3*.
  - **Notably**: 2 solutions for each angle were provided when each VLE was solved. This was because of the periodic nature of trig functions, which resulted in multiple solutions [2]. These had to be reordered to yield the plot in *Figure 3*.

Matlab Validation:

- All output angles were solved using input angles of 0 to 360 degrees on 1 degree intervals. As previously mentioned, the Solidworks ornithopter wings were able to execute continuous motion. This meant that all output angles, when plotted, should have had a continuous curve. This behavior can be seen in the plots in *Figure 3*.
- The solved for output Θ's were substituted into their original VLE's to validate that results were nearly zero. The answers for all solutions were within 10e-14 mm of 0.

Microsoft Excel Validation:

- Using the given angle input Θ_2 = 35°, Θ_3 and Θ_4 are determined to be 41.18° and 109.586° respectively. Since the x-axis position of 'A' and y-axis position of 'D' are 0, the positions of Bx, By, Cx, and Cy can be determined. Since 'A' and 'D' are fixed lengths with the driving link R2, determining initial angles of Θ_3 and Θ_4 will also result in the correct Θ_5,Θ_6 Θ_7,and Θ_8 angles.

Theta 2	Theta3	Theta4	Ax	Ay	Bx	By	Cx	Cy
32	42.6247	109.808	0	0	15.9433	9.96248	46.1113	37.7274113
33	42.1401	109.720	0	0	15.7670	10.2392	46.1687	37.7480408
34	41.6592	109.646	0	0	15.5859	10.5128	46.2174	37.7654799
35	41.1819	109.586	0	0	15.4000	10.7832	46.2575	37.7797758
36	40.7084	109.538	0	0	15.2095	11.0503	46.2890	37.7909718
37	40.2388	109.503	0	0	15.0143	11.3141	46.3120	37.7991076

Figure 4: Excel Table of Input/Output Angles and Positions

References

[1] “EQN.” *Equations and Systems Solver - MATLAB*,
https://www.mathworks.com/help/symbolic/sym.solve.html.
[2] *Trigonometric Equations*, https://xaktly.com/TrigonometricEquations.html.
[3] https://www.softintegration.com/chhtml/toolkit/mechanism/fourbar/fourbarpos.html

Appendices

Appendix A: Matlab Code to Make Figure 3

```
% Clearing all previous values and figures
clc
clear
close all
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Initializing 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;
% Input angle array. An array to account for ALL possible input angles.
theta2=linspace(0,361,361);
% Converting theta values to radians, which capture quadrant positions.
for angle = 1:length(theta2)
   theta2(angle) = theta2(angle)*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
for i = 1:length(theta2)
   msg = sprintf('Solving 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
            %disp("Alternative Solution for Correct Theta 3 & 4 Curve.")
            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
            %disp("Alternative Solution for Correct Theta 5 & 6 Curve.")
            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
            %disp("Alternative Solution for Correct Theta 7 & 8 Curve.")
            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.
% Initializing tolerance and failure counter
check_fails = 0;
tol = 0.00005; % Above or below zero for each VLE
% Looping through entire array
for angle = 2:length(theta2)
   % VLE 1 Check
   E1 =
R2*cos(theta2(angle))+R3*cos(theta3_arr(angle))-R4*cos(theta4_arr(angle))-R1*co
s(theta1);
   E2 =
R2*sin(theta2(angle))+R3*sin(theta3_arr(angle))-R4*sin(theta4_arr(angle))-R1*si
n(theta1);
   if abs(E1) > tol || abs(E2) > tol
       check_fails = check_fails + 1;
   end
   % VLE 2 Check
   theta_EC = theta3_arr(angle) + theta_EC_off; % cst
   theta_FC = theta4_arr(angle) + theta_FC_off; % cst
   E1 =
R_EC*cos(theta_EC)+R_FC*cos(theta_FC)-R6*cos(theta6_arr(angle))-R5*cos(theta5_a
rr(angle));
   E2 =
R_EC*sin(theta_EC)+R_FC*sin(theta_FC)-R6*sin(theta6_arr(angle))-R5*sin(theta5_a
rr(angle));
   if abs(E1) > tol || abs(E2) > tol
       check_fails = check_fails + 1;
   end
   % VLE 3 Check
   theta_GI = theta5_arr(angle); % Literally the same link.
```

```
   E1 =
R7*cos(theta7_arr(angle))-R8*cos(theta8_arr(angle))-R_GI*cos(theta_GI)+R6*cos(t
heta6_arr(angle));
   E2 =
R7*sin(theta7_arr(angle))-R8*sin(theta8_arr(angle))-R_GI*sin(theta_GI)+R6*sin(t
heta6_arr(angle));
   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
%%%%%%%%%%%%%%%%%%%
% Plotting Values
%%%%%%%%%%%%%%%%%%%
% Plotting all the output angles vs theta2
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\p
i'});
% Labeling title, x axis, & y axis.
title('Position Analysis','FontSize',20);
xlabel('\Theta2 (radians)','FontSize',18);
ylabel('Outputs (radians)','FontSize',18);
% Adding legends
legend = legend({'theta3','theta4','theta5','theta6','theta7','theta8'},...
   'Location','northwest','NumColumns',6);
legend.FontSize = 14;
% Modifying figure window size to automatically fit legend.
set(gcf, 'Position', [100,100,850,600])
```