Use of Computer Coding to Teach Design in a Mechanics Course, Resulting in an Implementation of a Kinematic Mechanism Design Tool Using PYTHON

Dr. Peter L. Schmidt PE, University of Evansville

Peter L. Schmidt received his bachelor's degree in mechanical engineering from the University of Louisville, a master's degree in mechanical engineering from the Rose-Hulman Institute of Technology and his doctorate degree in mechanical engineering from Vanderbilt University. He is currently an associate professor of Mechanical Engineering at the University of Evansville. He was previously appointed as an associate professor at the University of North Carolina at Charlotte. He has served as a research associate and as an instructor at Vanderbilt University. He has also worked at the Naval Surface Warfare Center in Crane, Indiana; at Precision Rubber, now part of Parker Hannifin in Lebanon, Tennessee; for CDAI in Atlanta, Georgia and at UTC / Carrier in Lewisburg, Tennessee. Dr. Schmidt is a member of the ASEE and a licensed professional engineer in Tennessee and Georgia. He is also a member of ASME, ASA and INCE. Dr. Schmidt's research interests include aeroacoustics and ultrasonics, and has authored several journal and conference papers on these subjects.

Mr. Philip Andrew Lax, University Of Evansville

Philip Lax is currently a 4th year Mechanical Engineering student at the University of Evansville, with minors in Mathematics and Chemistry. He is also a Student Trainee (Mechanical Engineer) at the Naval Surface Warfare Center, Crane Division.

Use of computer coding to teach design in a mechanics course, resulting in an implementation of a kinematic mechanism design tool using PYTHON

Abstract

Use of a computer project to teach design of simple mechanisms as a part of a traditional mechanisms course is discussed. Multiple software platforms were implemented, with sample output from each individual platform, including MATLAB source code is included in the results. Effect on student achievement in the course was compared to a realization prior to the implementation of the project. The work of a student who chose to implement the project using the PYTHON computer language is highlighted, with PYTHON code included in the work.

Introduction

Great emphasis has been placed on the importance of teaching students to code to support the need for computer literacy and to provide infrastructure for the computer based approach to problem solving made possible by advances in machine capability [1] [2].

Few studies have approached this situation from the opposite perspective. Does the process of coding a problem in some formalized way help a student to internalize the methodology itself and facilitate the conceptualization of a system, equation or procedure as an abstract design tool rather than just a use as a formulaic process that yields a single answer?

Deshpande, Waychal and Udawant [3] showed that programming activity improved design process performance when used in lieu of a portion of previously implemented lecture content.

Estrada and Aguiniga [4] have included use of computer coding in their course *Engineering Mechanics of Fiber Composites*, with notable improvement in student engagement resulting from reduced need for performing tedious hand calculations. This work was built on a previous study of a computation task, again with the goal of removing tedious hand calculations from the design learning process [5]. Others [6], [7] have reported similar results.

Brinson et al. [8] demonstrated the benefit of integrating programming into mechanics courses especially in the area of linear algebra use by students.

Kanive et al. [9] showed that the use of computers in mathematics instruction to novice learners was beneficial to student understanding and retention of methodology. Though it should be noted that this study was performed at grade school level, the concept remains applicable.

Deek et al. [10] examined the methodology for assessing this type of student work, concluding that subjective methodologies on process were useful in addition to strictly technical evaluation of the students' work product. This work guided the formation of the framework of the project instructions provided to the students for this study.

A junior course in traditional machine analysis was restructured to gather data on this question. The course is a traditional lecture, meeting three times a week for 50 minutes. This is an introductory course in machine analysis, with a prerequisite of a traditional Dynamics course, along with supporting mathematics courses. Assessments have traditionally been through homework assignments and in-class examinations, with homework structured as hand solutions to homework problems supplied with the text used in the course.

In fall of 2016, the homework portion of the assessment regime was restructured. Hand solutions were augmented with a semester long project to construct a design and analysis tool using a software platform of the students' choice. The goal was to compare the student's performance in the course to a cohort of students who completed the course with a more traditional assignment structure. This change was inspired, in part, by the positive response by students to the incorporation of the Freudenstein Equation [11] as a design tool for four-bar linkages during the 2015 offering of this course.

The basic lecture topical order was:

1. Kinematic analysis of linkages
2. Kinetic analysis of linkages
3. Cam design
4. Cam kinematic analysis
5. Cam kinetic analysis, including vibration performance
6. Balancing of linkages

It should be noted that the text used with the course is supplied with code for accomplishing the calculations required in performance of the project [12]. This code was not referenced in lecture, but the students did have access to this information via an electronic supplement supplied with their texts.

Design and analysis tool

The students were provided with the basic requirements for the performance of their software tool, and provided guidance on the basic structure of the user interface. The following direction was provided:

Your semester project will be to create a software design tool capable of analyzing 4 bar linkages, crank slider mechanisms and cam – follower systems.

You may use either a spread sheet (MS Excel®) or computer code (MATLAB® or OCTAVE) to accomplish this.

Your tool will interrogate a user for known inputs and supply calculated outputs for the following types of problems:

1. *Position of a 4 bar linkage*
2. *Position of a crank slider*
3. *Velocity of a 4 bar linkage*
4. *Velocity of a crank slider*
5. *Acceleration of a 4 bar linkage*
6. *Acceleration of a crank slider.*
7. *Forces on a 4 bar linkage*
8. *Forces on a crank slider*
9. *Cam and follower position, velocity, acceleration and jerk.*
10. *Vibration of cam follower systems.*

Each numbered item above will be accomplished using a separate spread sheet tab or a separate MATLAB / OCTAVE subroutine. Any modality will contain notation that makes it obvious to the user which quantities need to be entered and which quantities are solution results. You will use worked problems given in your text as verification examples for your work.

Students were supplied with an intentionally incomplete hard copy example spreadsheet to assist them with formulation of a strategy for organization of their work. MATLAB [13] code was suggested as the vehicle for programming due to its widespread use and availability in academia, along with the fact that OCTAVE software is available as a design tool for all practitioners. The availability of example code [14] from the MATLAB programming community also contributed to this choice.

Homework assignments were structured with both hand solutions and machine solutions required for a set of assigned problems. The solutions required did not include all problems and all methods, but rather a set of discrete hand solutions and a set of discrete machine solutions. This approach was chosen so that the students could discover, or not, that the machine tool was useful in checking their hand solutions and that the hand solutions were useful in checking their machine code.

Student performance

Restructuring of the homework assignments had a positive effect on the performance of the students in the 2016 cohort. Examination average scores were higher at the beginning of the course, as shown in Figure 1, when kinematic analysis was being stressed in lecture and in assignments. It is interesting to note that the scores dipped a bit when the lecture material transitioned to discussion of cam design, which represented a new design concept as compared to the analysis of linkages, which could be related to work done in previous mechanics courses.

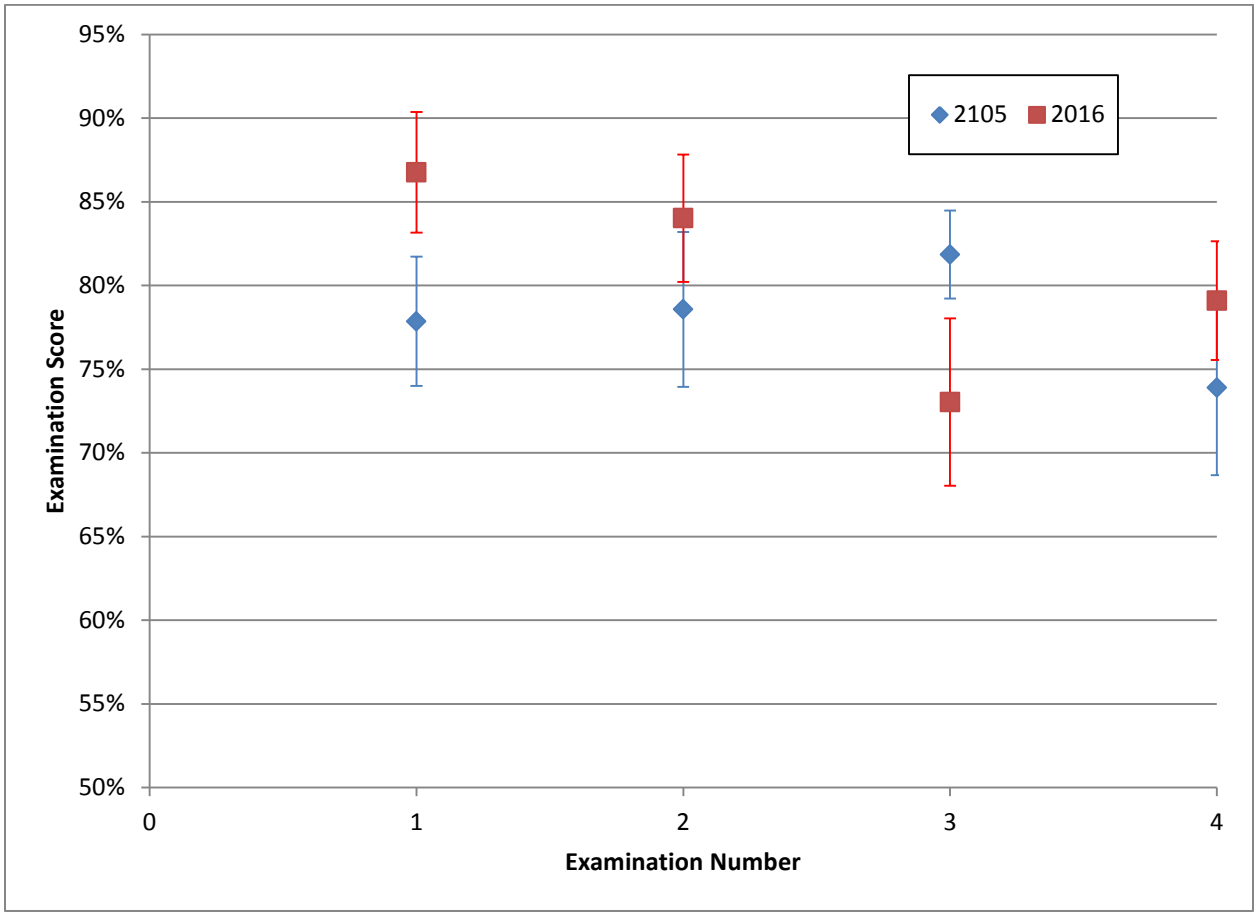

Figure 1. Examination grades for two cohorts of students. The student cohorts were of comparable size, n = 28 for 2015, n = 34 for 2016. The error bars show a standard error based on a 95% confidence interval.

The error bars shown in Figure 1 represent the standard error based on a 95% confidence interval and are calculated using Equation (1). Standard error (u_r) is given by

$$u_r = \frac{k\sigma}{\sqrt{N}} \tag{1}$$

where σ is the standard deviation of the population, N is the population size and k is a constant determined by the desired confidence interval. In this instance $k = 2$ for the 95% confidence interval stated.

Homework scores displayed two interesting trends. Figure 2 shows a comparison between the two cohorts based on scores for submitted items only, i.e., non-submittal scores of zero were omitted from the average scores shown. Again, the dip in scores around the time cam design was introduced in lecture is present. Again, the error bars shown in Figure 2 represent the standard error based on a 95% confidence interval and are calculated using Equation (1).

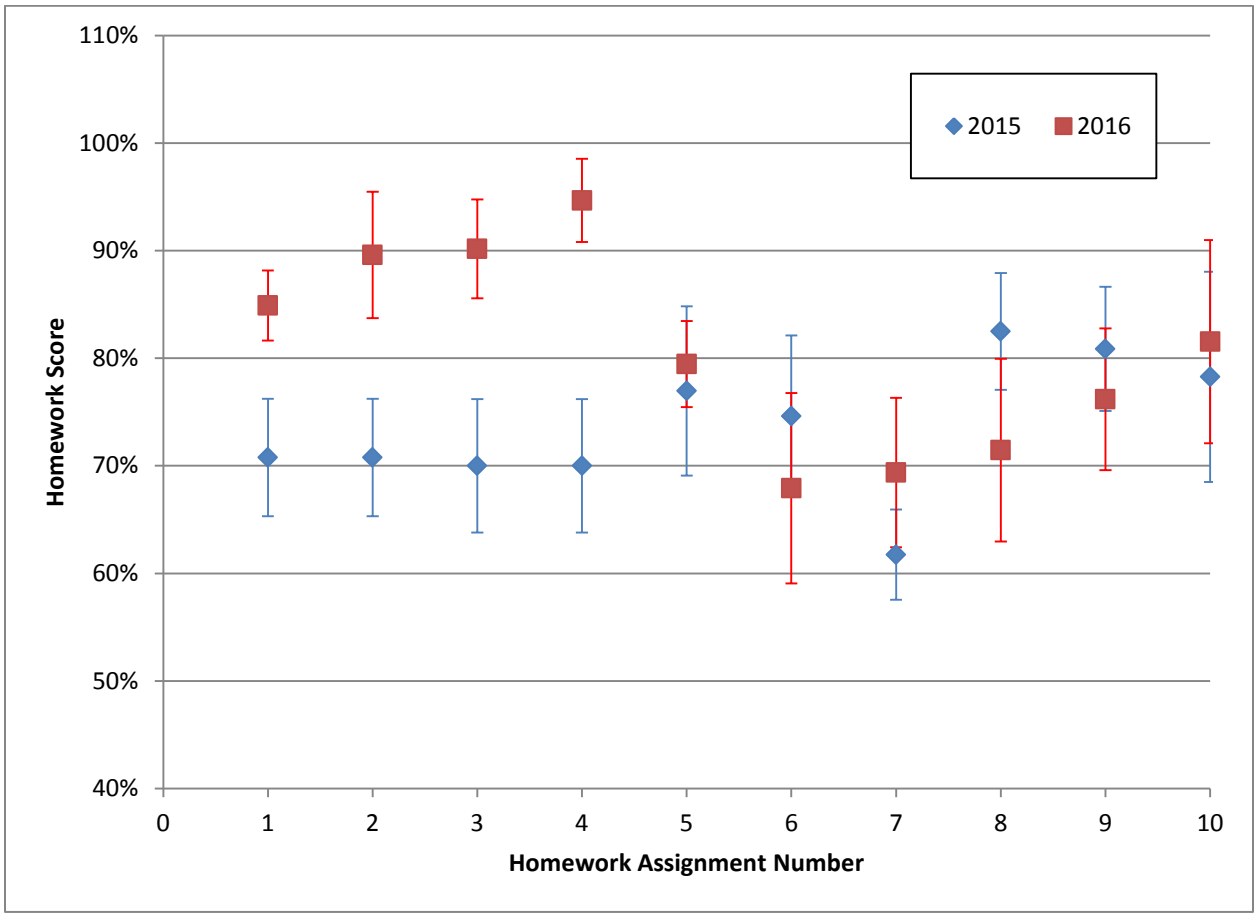

Figure 2. Average Homework Scores for two cohorts of students. The error bars show a standard error based on a 95% confidence interval.

Figure 3 illustrates an interesting effect that the restructured assignment format had on homework submittal practice by the students. The data are the individual scores for each homework assignment turned in by the entire cohort of students. The only scores of 0% were

assigned for students who did not submit an individual assignment. The histogram shows that the number of non-attempts for homework assignments rose when the new structure was instituted. This could be due to the lack of availability of solutions for such problems or could be attributed to procrastination on the part of students who viewed the project as a unitary submittal due at the end of the semester, rather than as a tool for learning to be developed along with their lecture notes.

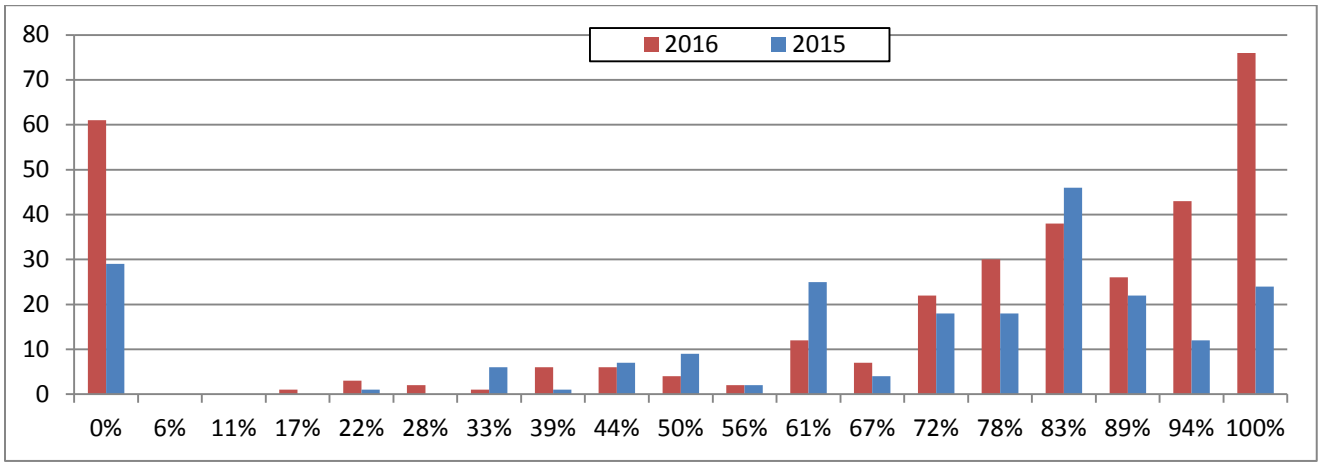

Figure 3. Histogram of homework grades for two cohorts of students.

Figure 4 also shows that the quality of actual attempts improved with the new assignment structure. Attempts in the top decade of scores were at a maximum for the 2016 cohort, while the maximum for the 2015 cohort fell around the 83rd percentile. This effect was also present for exam scores, as shown in Figure 5.

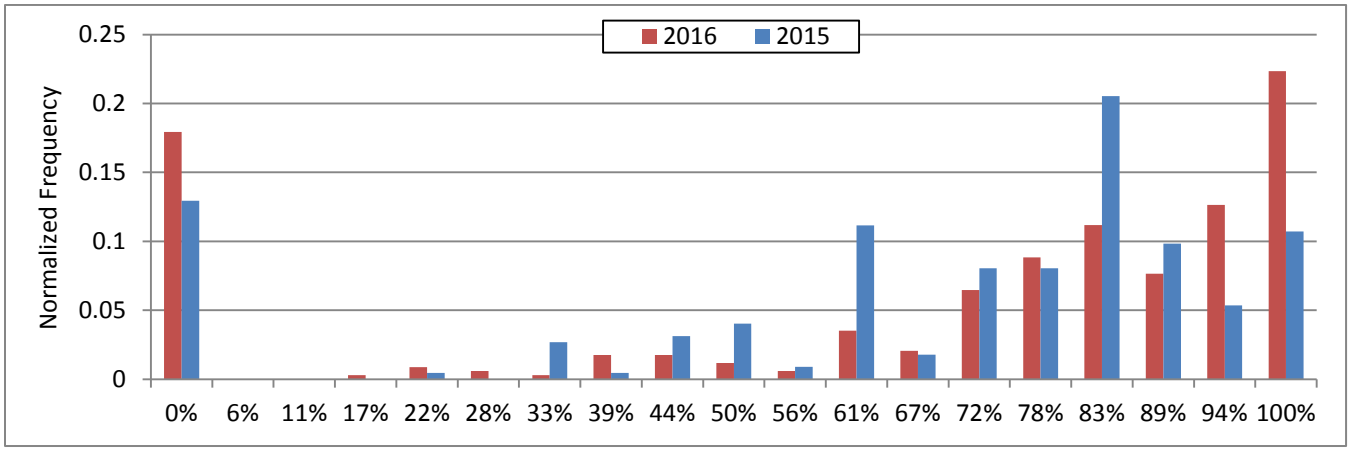

Figure 4. Histogram of homework grades for two cohorts of students, normalized to sample size.

Scores in the B and D range dropped, with a concomitant rise in A level work.

Again, the data shown in Figure 5 are the individual scores on each examination for the entire cohort of students.

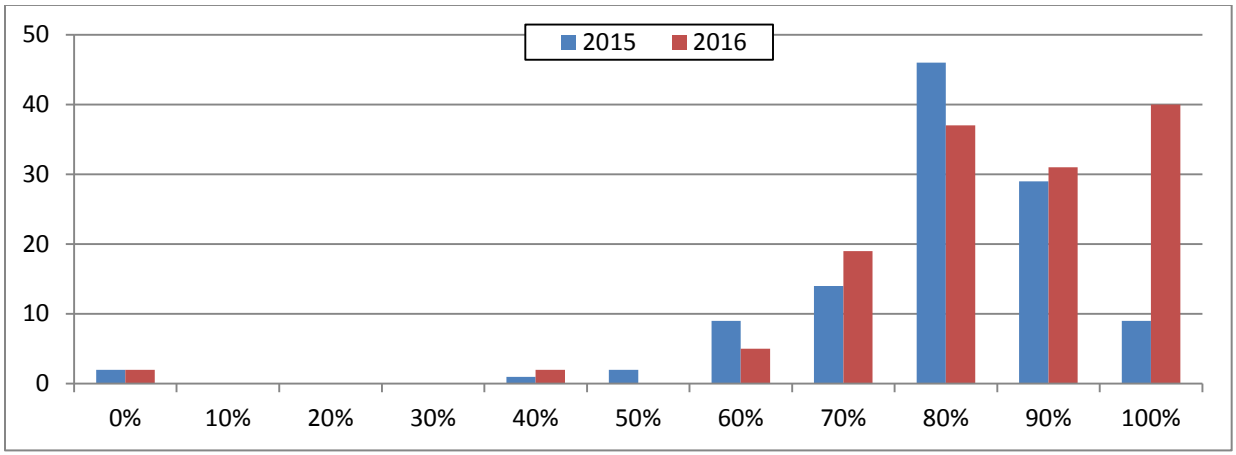

Figure 5. Histogram of examination scores for two cohorts of students.

Note that the 2015 cohort had two visiting international students who were not on a normal track to graduate in the program. The 2016 cohort had a single international student who abandoned the course and was unable to drop due to restrictions from the funding agency supporting that student.

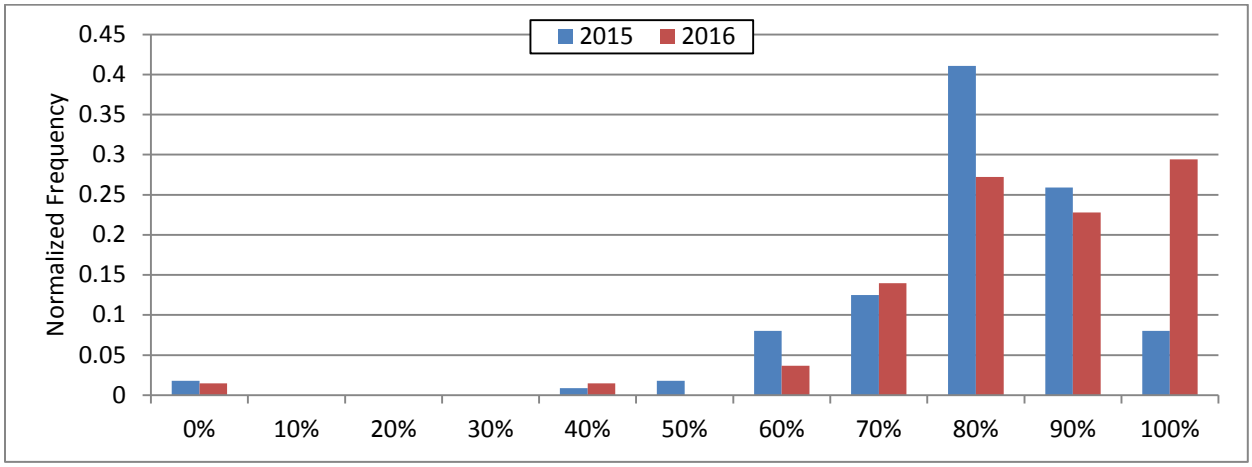

Figure 6. Histogram of examination scores for two cohorts of students, normalized to sample size.

Figure 6 shows the same decrease in B and D level grades, along with the associated increase in A level grades shown in Figure 4.

Fig 7 shows the distribution of the students' approach to the project, with the largest subset of the students following the structure provided using a spreadsheet approach rather than viewing it as

an example possibility. Note that some students submitted both MATLAB and EXCEL® work product.

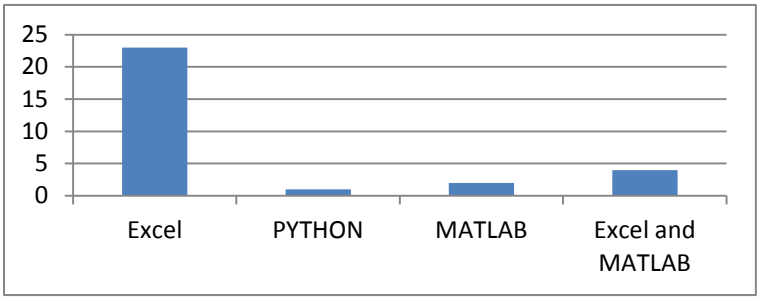

Figure 7. Student submittal types

Example student realizations / spread sheet form

The students were instructed to keep the standard notation for four bar linkages shown in the lecture notes. Figure 8 shows the numbering conventions for a typical linkage, shown in the open position [12]. A crossed or closed configuration would be realized with all links remaining at the same lengths shown in Figure 6, with link 4 reflected across the horizontal axis with link 3 still connecting the ends of links 2 and 4.

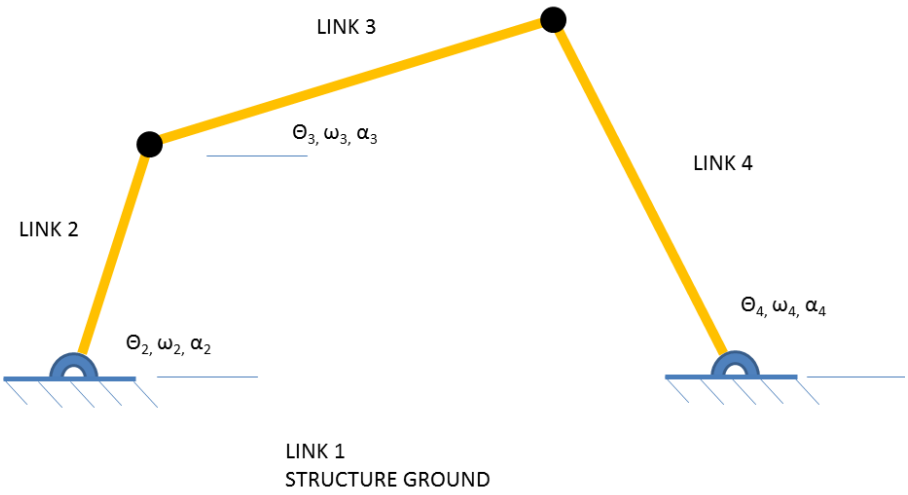

Figure 8. Link, angle, angular velocity and angular acceleration numbering conventions used in student work product.

As shown in Figure 9, the typical realization was a tabbed spreadsheet executed in MS Excel® with minimal formatting and organization mimicked from the sample provided to the students. Many students struggled to label inputs and outputs of the methodology they coded. Figure 9 shows an example of average student work product. The work product was typically poorly documented, with minimal labeling of displayed quantities, making it of limited utility to a user not involved directly with the lecture series or to a user who was not intimately involved in the creation of the spreadsheet.

ω2	δ3	R_{pa}	Θ_{31}	Θ_{32}	Θ_{41}	Θ_{42}
1	1	1	0	277.5118704	57.3	220.2118704
OPEN	**OPEN**	**OPEN**	**OPEN**	**OPEN**		
ω31 (rad/sec)	ω41 (rad/sec)	VA_x	VA_y	VA		
0	1	-42.0755391	27.012	50		
VB_x	VB_y	VB				
-42.0755391	27.01201602	50				
CLOSED	**CLOSED**	**CLOSED**	**CLOSED**	**CLOSED**		
ω32(rad/sec)	ω42 (rad/sec)	VA_x	VA_y	VA		
0.232789491	-0.767210509	-42.0755391	27.012	50		
VB_x	VB_y	VB				
32.28079575	-38.1831144	50				

Figure 9. Typical student submission of spreadsheet form.

Figure 10 shows a more sophisticated spreadsheet submittal. Note proper labeling of the input and output fields. Both of these sets of calculations refer to a four bar linkage as shown in the course text. This submittal also illustrates poor documentation of the method, with no equation supplied to the user in easily viewable form. Students were counseled on multiple occasions to supply equations entered in text boxes, rather than requiring that the user decipher the spreadsheet code. The cells containing references to a quantity of V, with some subscript, refer to velocities of centers of mass of individual links, used in reaction force calculations, or relative velocities of two points used in calculation of angular velocity. It is also interesting to note that this student took the time to properly subscript the labels for displayed quantities.

Input Data										
a	b	c	θ_2	θ_{31}	θ_{32}	θ_{41}	θ_{42}	R_{PA}	δ	ω_2
50	75	50	57.29578	0	-82.4891	57.29578	-139.785	0	0	1

Output Data						
ω_{31}	ω_{32}	ω_{41}	ω_{42}	$\lvert V_A \rvert$	V_{AX}	V_{AY}
0.000	0.233	1.000	-0.767	50	-42.0735	27.01512

$\lvert V_{B1} \rvert$	V_{B1X}	V_{B1Y}	$\lvert V_{B2} \rvert$	V_{B2X}	V_{B2Y}
50	-42.0735	27.01512	38.3649	-24.7706	29.29643

$\lvert V_{BA1} \rvert$	V_{BA1X}	V_{BA1Y}	$\lvert V_{BA2} \rvert$	V_{BA2X}	V_{BA2Y}
1.47E-14	0	1.47E-14	17.45264	17.3029	2.281315

$\lvert V_{PA1} \rvert$	V_{PA1X}	V_{PA1Y}	$\lvert V_{PA2} \rvert$	V_{PA2X}	V_{PA2Y}
0	0	0	0	0	0

$\lvert V_{P1} \rvert$	V_{P1X}	V_{P1Y}	$\lvert V_{P2} \rvert$	V_{P2X}	V_{P2Y}
50	-42.0735	27.01512	50	-42.0735	27.01512

Indicates Data Pulled from a Previous Tab

Figure 10. Properly formatted submission of a spreadsheet realization.

Example student realizations / Interpreted script form

7 students of 30 in the cohort attempted to produce script files to accomplish the semester project. Of these, all save one attempted to use the computation console detailed in the project instructions, the OCTAVE / MATLAB environment. An example block of MATLAB code is supplied in Appendix 1.

One student chose to use a computing environment mastered outside the Mechanical Engineering curriculum as a development tool for producing MATLAB code. This student demonstrated the code during office hours as a part of a discussion among a few faculty and students. The code was partially complete in a numerical sense (items 1 – 6 on the list provided in the assignment instructions were complete) but had been augmented with animation of mechanisms using tools available in the PYTHON computing environment.

This demonstration led to the student being given permission to pursue the project in the PYTHON language in order to encourage additional capability in the code and to reward initiative.

The complete PYTHON code for this realization is included as Appendices 2 and 3. A screen shot of the animation routine is shown in Figure 11. Note that link 3 in this mechanism is shown with more complex geometry that that shown in Figure 8, which was included in the study of forces on link members when in motion.

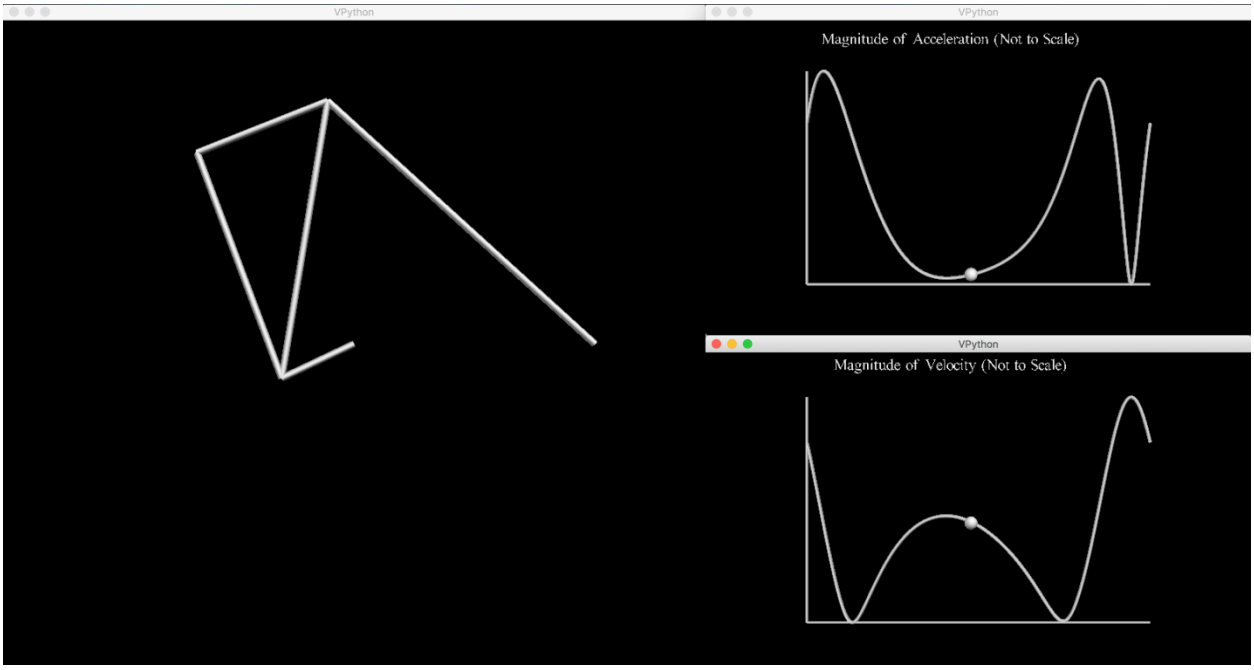

Figure 11. Screenshot of the Python code animation realization of the project.

Conclusion

Data on student performance improvements due to institution of a programming project in lieu of traditional homework solution generation has been presented. Use of the project structure has been shown to increase student performance on homework assignments and on examination scores in the aggregate. Data also shows that the student performance has exhibited a shifted distribution, with peak performance falling above the 90^{th} percentile. The programming project has also increased the number of non-submittals of homework assignments by students.

References

[1] Esther Shein, "Should Everybody Learn to Code?," *Communications of the Association for Computing Machinery*, vol. 57, no. 2, pp. 16-18, February 2014.

[2] Philip Guo, "Teaching Programming the Way It Works Outside the Classroom," *Communications of the Association of Computing Machinery*, vol. 56, no. 8, pp. 10-11, August 2013.

[3] Manojkumar Deshpande, Pradeep Waychal, and Prashant Udawant, "Analysis of Improved Pedagogy Applied for Teaching courses related to Computer Programming for First Year Engineering Programs," in *ASEE International Forum*, Seattle, 2015.

[4] H. Estrada and F. Aguiniga, "Analysis of Laminated Composites: A Web-based Computer Program Based on Classical Lamination Theory," in *American Society for Engineering Education Annual Conference & Exposition*, Chicago, 2005.

[5] H. Estrada and Y. Chiu, "Analysis of Wind Loads on Buildings and Signs: A Computer Program Based on ASCE 7," in *American Society for Engineering Education Annual Conference & Exposition*, Portland, Oregon, 2004.

[6] Christopher Hundhausen, Pawan Agarwal, Richard Zollars, and Adam Carter, "The Design and Experimental Evaluation of a Scaffolded Software Environment to Improve Engineering Students' Disciplinary Problem-Solving Skills," *Journal of Engineering Education*, vol. 100, no. 3, pp. 574-603, July 2011.

[7] Elliot Soloway, "Learning to Program = Learning to Construct Mechanisms and Explainations," *Communications of the Association of Computing Machinery*, vol. 29, no. 9, pp. 850-858, September 1986.

[8] L. Cate Brinson, Ted Belytschko, Brian Moran, and Tom Black, "Design and Computational Methods in Basic Mechanics Courses," *Journal of Engineering Education*, vol. 22, no. 4, pp. 159-166, April 1997.

[9] Rebecca Kanive, Peter Nelson, and Matthew, Ysseldyke, James Burns, "Comparison of the Effects of Computer-Based Practice and Conceptual Understanding Interventions on Mathematics Fact Retention and Generalization," *Journal of Educational Research*, vol. 107, pp. 83-89, 2014.

[10] Fadi Deek, Starr Roxanne Hiltz, and Howard, Rotter, Naomi Kimmel, "Cognitive Assessment of Students' Problem Solving and Program Development Skills," *Journal of Engineering Education*, vol. 88, no. 4, pp. 317-326, July 1999.

[11] Ferdinand Freudenstein, "Approximate Synthesis of Four-Bar Linkages," *Transactions of the American Society of Mechanical Engineers*, vol. 77, pp. 853-861, August 1955.

[12] Robert Norton, *Design of Machinery*, 5th ed. New York, United States of America: McGraw Hill, 2012.

[13] Mathworks Incorporated. (2016, August) The MathWorks. [Online]. https://www.mathworks.com/

[14] Ahmad Smaili and Firas Zeineddine, "SoftLink: A Matlab/Simulink Based Code for the Analysis, Synthesis, Optimization and Simulation of Mechanisms," in *American Society for Engineering Education Annual Conference & Exposition*, Nashville, 2003.

Appendix 1

MATLAB / OCTAVE code for creation and analysis of a cam profile.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
%Program Name: CamAnalysis
%
%Program Description: Analyzes and Creates Cam Profile
%
%Inputs:              Number of Zones and the Parameters
associated with
%                     each
%
%Outputs:             S,V,A,J Curves, Force, Power, Torque,
Pressure Angle,
%                     and Cam Profile Plots. Tabular Data Sets.
Max Values.
%
%Date Created:        11-5-2016
%
%Revisions:
%
%0)    11-5-2016    Creation
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%


clear
clc

%Parameters%

s_harmonic = @(h,theta,Beta,Beta_time) h/2*(1-
cos(pi*theta/Beta));
v_harmonic = @(h,theta,Beta,Beta_time)
pi*h/2/Beta_time*sin(pi*theta/Beta);
a_harmonic = @(h,theta,Beta,Beta_time)
pi^2*h/2/Beta_time^2*cos(pi*theta/Beta);
j_harmonic = @(h,theta,Beta,Beta_time) -
pi^3*h/2/Beta_time^3*sin(pi*theta/Beta);

s_cycloid = @(h,theta,Beta,Beta_time) h*(theta/Beta -
1/2/pi*sin(2*pi*theta/Beta));
v_cycloid = @(h,theta,Beta,Beta_time) h/Beta_time*(1-
cos(2*pi*theta/Beta));
```

```
a_cycloid = @(h,theta,Beta,Beta_time)
2*pi*h/Beta_time^2*sin(2*pi*theta/Beta);
j_cycloid = @(h,theta,Beta,Beta_time)
4*pi^2*h/Beta_time^3*cos(2*pi*theta/Beta);

tstep = .02;
%Inputs% If you want to actually input values, uncomment lines
21 through
%35

% zone_numbers = input('Please input the number of zones:');

% for n=1:zone_numbers
%   zone_type(n) = input('Please input the type of each zone: 1
for Rise, 2 for Dwell, or 3 for Fall: ');
%   zone_mag(n)  = input('Please input the magnitude of the rise
(+) or fall (-) (0 for dwells): ');
%   zone_model(n)= input('Please input the zone model type: 0
for Dwell, 1 for Harmonic, 2 for Cycloidal: ');
%   zone_time(n) = input('Please input the time for each
section: ');
% end
% circleB = input('Please input the radius of the base circle:
');
% follower_rad = input('Please input the radius of the follower:
');
% follower_mass = input('Please input the mass of the follower:
');
% follower_spring_stiff = input('Please input the spring
stiffness: ');
% damping = input('Please input the damping ratio: ');
% preload = input('Please input the preload on the follower (0
for none): ');

zone_numbers = 4;
zone_type = [1 2 3 2];
zone_mag = [.025 0 -.025 0];%meters Needs to be in meters for
Newtons later on
zone_model = [1 0 2 0];
zone_time = [.75 1.25 .5 .25];
circleB = .087;%meters
follower_rad = .012;%meters
follower_mass = 2;
follower_spring_stiff = 35*1000;%N/m
damping = .045;
preload = 5;
```

```
%Workspace%

period = sum(zone_time);
tcount = 1;
omega = 2*pi/period;

time(tcount) = 0;
theta_array(tcount)= 0;
s_array(tcount) = circleB;
v_array(tcount) = 0;
a_array(tcount) = 0;
j_array(tcount) = 0;

for n=1:zone_numbers
  beta_array(n) = zone_time(n)/period*2*pi;
end

for n = 1:zone_numbers
  if zone_model(n) == 0
    fprintf('\nZone %1.1f\n=======\n\n',n)
    s = 0
    v = 0
    a = 0
    j = 0
    for t = 0:tstep:zone_time(n)
      tcount = tcount+1;
      time(tcount) = time(tcount - 1) + tstep;
      theta_array(tcount) = theta_array(tcount-1)+omega*tstep;
      s_array(tcount) = s_array(tcount-1);
      v_array(tcount) = 0;
      a_array(tcount) = 0;
      j_array(tcount) = 0;
    end
  end
  if zone_model(n)==1
    fprintf('\nZone %1.1f\n========\n\n',n)
    h = zone_mag(n);
    Beta = beta_array(n);
    Beta_time = zone_time(n);
    s_old = s_array(tcount);
    v_old = v_array(tcount);
    a_old = a_array(tcount);
    j_old = j_array(tcount);
    fprintf('s = %5.2f(1-cos(%5.2f*theta))\n',h/2,pi/Beta)
    fprintf('v = %5.2f(sin(%5.2f*theta))\n',
pi*h/Beta_time/2,pi/Beta)
```

```
    fprintf('a = %5.2f(cos(%5.2f*theta))\n',
pi^2*h/Beta_time^2/2,pi/Beta)
    fprintf('j = %5.2f(sin(%5.2f*theta))\n', -
pi^3*h/Beta_time^3/2,pi/Beta)

    for t = 0:tstep:zone_time(n)
      tcount = tcount+1;
      time(tcount) = time(tcount - 1) + tstep;
      theta = omega*t;
      theta_array(tcount) = theta_array(tcount-1)+omega*tstep;
      s_array(tcount) =
s_harmonic(h,theta,Beta,zone_time(n))+s_old;
      v_array(tcount) =
v_harmonic(h,theta,Beta,zone_time(n))+v_old;
      a_array(tcount) =
a_harmonic(h,theta,Beta,zone_time(n))+a_old;
      j_array(tcount) =
j_harmonic(h,theta,Beta,zone_time(n))+j_old;
    end
  end
  if zone_model(n) == 2
    fprintf('\nZone %1.1f\n========\n\n',n)
    h = zone_mag(n);
    Beta = beta_array(n);
    Beta_time = zone_time(n);
    s_old = s_array(tcount);
    v_old = v_array(tcount);
    a_old = a_array(tcount);
    j_old = j_array(tcount);
    fprintf('s = %5.2f(theta/%5.2f-
%5.2fcos(%5.2f*theta))\n',h,Beta,1/2/pi,2*pi/Beta)
    fprintf('v = %5.2f(1-cos(%5.2f*theta))\n',
h/Beta_time,2*pi/Beta)
    fprintf('v = %5.2fsin(%5.2f*theta)\n',
2*pi*h/Beta_time^2,2*pi/Beta)
    fprintf('v = %5.2fsin(%5.2f*theta)\n',
4*pi^2*h/Beta_time^3,2*pi/Beta)

    for t = 0:tstep:zone_time(n)
      tcount = tcount+1;
      time(tcount) = time(tcount - 1) + tstep;
      theta = omega*t;
      theta_array(tcount) = theta_array(tcount-1)+omega*tstep;
      s_array(tcount) =
s_cycloid(h,theta,Beta,zone_time(n))+s_old;
      v_array(tcount) =
v_cycloid(h,theta,Beta,zone_time(n))+v_old;
```

```
      a_array(tcount) =
a_cycloid(h,theta,Beta,zone_time(n))+a_old;
      j_array(tcount) =
j_cycloid(h,theta,Beta,zone_time(n))+j_old;
    end
  end
end

pressure_angle_array = (v_array./omega)./(s_array +
sqrt((circleB+follower_rad)^2));
force_array =
follower_mass*(a_array*follower_mass+damping*v_array+follower_sp
ring_stiff*(s_array-circleB))+preload;
follower_force = force_array.*cos(pressure_angle_array);
torque_array = follower_force.*v_array/omega;
power_loss_array =
abs(force_array.*cos(pressure_angle_array).*v_array);

s_max = max(s_array);
v_max = max(v_array);
a_max = max(a_array);
j_max = max(j_array);
phi_max = max(pressure_angle_array);
f_max = max(force_array);
t_max = max(torque_array);
p_max = max(power_loss_array);

fprintf('\nThe angular velocity of the cam is: %5.2f
radians/sec',omega)
fprintf('\nThe maximum displacement is %2.2f inches',s_max-
circleB)
fprintf('\nThe maximum velocity is %2.2f inches/second',v_max)
fprintf('\nThe maximum acceleration is %2.2f
inches/second^2',a_max)
fprintf('\nThe maximum jerk is %2.2f inches/second^3',j_max)
fprintf('\nThe maximum pressure angle is: %5.2f
degrees',phi_max*180/pi)
fprintf('\nThe maximum force is %5.2f lbf',f_max)
fprintf('\nThe maximum torque is %5.2f inlbs', t_max)
fprintf('\nThe maximum power loss is %5.2f inlbs/sec', p_max)

figure
plot(time,s_array-circleB)
xlabel('Time (s)')
ylabel('Displacement (in)')
title('Displacement vs time')
```

```
figure
plot(time,v_array)
xlabel('Time (s)')
ylabel('Velocity (in/s)')
title('Velocity vs time')

figure
plot(time,a_array)
xlabel('Time (s)')
ylabel('Acceleration (in/s^2)')
title('Acceleration vs time')

figure
plot(time,j_array)
xlabel('Time (s)')
ylabel('Jerk (in/s^3)')
title('Jerk vs time')

figure
plot(time,pressure_angle_array)
xlabel('Time (s)')
ylabel('Pressure Angle (radians)')
title('Pressure Angle vs time')

figure
plot(time, force_array)
xlabel('Time (s)')
ylabel('Cam Force (lbf)')
title('Cam Force vs time')

figure
plot(time, torque_array)
xlabel('Time (s)')
ylabel('Torque (inlbf)')
title('Torque vs time')

figure
plot(time, power_loss_array)
xlabel('Time (s)')
ylabel('Power (inlbf/s)')
title('Power vs time')

figure
% polar(theta_array,s_array)%plots cam profile for Octave Users
polarplot(theta_array,s_array)% Plots cam profile  For MATLAB
Users
```

```
data = [time' theta_array' s_array' v_array' a_array' j_array'
pressure_angle_array' force_array' torque_array'
power_loss_array'];
fprintf('\n   Time        Angle      Disp       Vel        Accel
Jerk       Phi        Force      Torque
Power\n=============================================================
=============================================\n')
disp(data)
```

Appendix 2

Python code submitted for realization of the student project described in this work.

```
from __future__ import division
from visual import *

d = 6 #link 1 length
a = 2 #link 2 length
b = 7 #link 3 length
c = 9 #link 4 length
theta2deg = 30 #theta2 (degrees)
omega2 = 10 #omega2 (rad/sec)
alpha2 = 0 #alpha2 (rad/sec**2)
rp = 6 #Rpa (arbitrary length)
delta3deg = 30 #delta3 (deg)
config = 1 #open (1) or crossed (2) configuration
theta2 = theta2deg*(pi/180)
delta3 = delta3deg*(pi/180)

deltatheta2 = pi/500

print "d = ",d
print "a = ",a
print "b = ",b
print "c = ",c
print "theta2 = ",theta2deg
print "omega2 = ",omega2
print "alpha2 = ",alpha2
print "rpa = ",rp
print "delta3 = \n",delta3deg

#adjust display size here using screen resolution
xdim = 1600
ydim = 900

scene = display(x=0, y=0, width=xdim*(900/1600),
height=ydim*(850/900))

#**WORKSPACE**

def Loop(a,b,c,d,theta2,omega2,alpha2,rp,delta3,config):
```

```
    #Vector Loop equations substituion values begin here
    k1 = d/a
    k2 = d/c
    k3 = (a**2-b**2+c**2+d**2)/(2*a*c)
    k4 = d/b
    k5 = (c**2-d**2-a**2-b**2)/(2*a*b)

    A = cos(theta2)-k1-k2*cos(theta2)+k3
    B = -2*sin(theta2)
    C = k1-(k2+1)*cos(theta2)+k3
    D = cos(theta2)-k1+k4*cos(theta2)+k5
    E = -2*sin(theta2)
    F = k1+(k4-1)*cos(theta2)+k5

    #Vector Loop equations substitution values have ended

    n31 = -E-sqrt(E**2-4*D*F) #The open angle theta3 numerator
is calculated
    n32 = -E+sqrt(E**2-4*D*F) #The crossed angle theta3
numerator is calculated
    d3 = 2*D #The denominator for the theta3 value is caclulated

    n41 = -B-sqrt(B**2-4*A*C) #The open angle theta4 numerator
is calculated
    n42 = -B+sqrt(B**2-4*A*C) #The crossed ange theta4 numerator
is calculated
    d4 = 2*A #The denominator for the theta4 value is caclulated

    #if the denominator is zero, print error messsage
    #if else, calculate values of theta3 (degrees) using Vector
Loop equation
    if d3 == 0:
        disp("you are dividing by zero in your theta3 arctan
function")
    else:
        theta31 = 2*atan(n31/d3)
        theta32 = 2*atan(n32/d3)

    #if the denominator is zero, print error messsage
```

```
    #if else, calculate values of theta4 (degrees) using Vector
Loop equation
    if d4 == 0:
        disp("you are dividing by zero in your theta4 arctan
function")
        theta41 = pi
        theta42 = pi
    else:
        theta41 = 2*atan(n41/d4)
        theta42 = 2*atan(n42/d4)

    #calculates values for open and crossed omega 3 (rad/time)
    omega31 = (a*omega2*sin(theta41-theta2))/(b*sin(theta31-
theta41))
    omega32 = (a*omega2*sin(theta42-theta2))/(b*sin(theta32-
theta42))

    #calculates values for open and crossed omega 4 (rad/time)
    omega41 = (a*omega2*sin(theta2-theta31))/(c*sin(theta41-
theta31))
    omega42 = (a*omega2*sin(theta2-theta32))/(c*sin(theta42-
theta32))

    #calculates value of the magnitude of va (length/time)
    vax = -a*omega2*sin(theta2)
    vay = a*omega2*cos(theta2)
    va = sqrt(vax**2+vay**2)

    thetaa = atan2(vay,vax)

    #calculates value of the magnitude of vb1 (length/time)
    vbx1 = -c*omega41*sin(theta41)
    vby1 = c*omega41*cos(theta41)
    vb1 = sqrt(vbx1**2+vby1**2)

    #calculates value of the direction of vb1 (deg)
    thetab1 = atan2(vby1,vbx1)

    #calculates value of the magnitude of vb2 (length/time)
    vbx2 = -c*omega42*sin(theta42)
    vby2 = c*omega42*cos(theta42)
```

```
    vb2 = sqrt(vbx2**2+vby2**2)

    #calculates value of the direction of vb1 (deg)
    thetab2 = atan2(vby2,vbx2)

    #calculates value of the magnitude of vp1 (length/time)
    vpax1 = -rp*omega31*sin(theta31+delta3)
    vpay1 = rp*omega31*cos(theta31+delta3)
    vpx1 = vpax1 + vax
    vpy1 = vpay1 + vay
    vp1 = sqrt(vpx1**2+vpy1**2)

    #calculates value of the direction of vp1 (deg)
    thetap1 = atan2(vpy1,vpx1)

    #calculates value of the magnitude of vp2 (length/time)
    vpax2 = -rp*omega32*sin(theta32+delta3)
    vpay2 = rp*omega32*cos(theta32+delta3)
    vpx2 = vpax2 + vax
    vpy2 = vpay2 + vay
    vp2 = sqrt(vpx2**2+vpy2**2)

    #calculates value of the direction of vp2 (deg)
    thetap2 = atan2(vpy2,vpx2)

    def
Accel(a,b,c,theta2,theta3,theta4,omega2,omega3,omega4,alpha2,rp,
delta3):

        A = c*sin(theta4)
        B = b*sin(theta3)
        C =
a*alpha2*sin(theta2)+a*omega2**2*cos(theta2)+b*omega3**2*cos(the
ta3)-c*omega4**2*cos(theta4)
        D = c*cos(theta4)
        E = b*cos(theta3)
        F = a*alpha2*cos(theta2)-a*omega2**2*sin(theta2)-
b*omega3**2*sin(theta3)+c*omega4**2*sin(theta4)

        alpha3 = (C*D-A*F)/(A*E-B*D)
        alpha4 = (C*E-B*F)/(A*E-B*D)
```

```
        AcAx = -a*alpha2*sin(theta2)-a*omega2**2*cos(theta2)
        AcAy = a*alpha2*cos(theta2)-a*omega2**2*sin(theta2)

        AcA = sqrt(AcAx**2+AcAy**2)
        thetaAcA = atan2(AcAy,AcAx)

        AcBAx = -b*alpha3*sin(theta3)-b*omega3**2*cos(theta3)
        AcBAy = b*alpha3*cos(theta3)-b*omega3**2*sin(theta3)

        AcBA = sqrt(AcBAx**2+AcBAy**2)
        thetaAcBA = atan2(AcBAy,AcBAx)

        AcBx = -c*alpha4*sin(theta4)-c*omega4**2*cos(theta4)
        AcBy = c*alpha4*cos(theta4)-c*omega4**2*sin(theta4)

        AcB = sqrt(AcBx**2+AcBy**2)
        thetaAcB = atan2(AcBy,AcBx)

        AccelPA =
cross(vector(0,0,alpha3),vector(cos(theta3+delta3)*rp,sin(theta3
+delta3)*rp,0))-
omega3**2*vector(cos(theta3+delta3)*rp,sin(theta3+delta3)*rp,0)
        AccelP = AccelPA + vector(AcAx,AcAy,0)
        AcP = mag(AccelP)
        thetaAcP = atan2(AccelP.y,AccelP.x)

        return
alpha3,alpha4,AcA,thetaAcA,AcBA,thetaAcBA,AcB,thetaAcB,AcP,theta
AcP


    x =
Accel(a,b,c,theta2,theta31,theta41,omega2,omega31,omega41,alpha2
,rp,delta3)

    alpha31 = x[0]
    alpha41 = x[1]
    AcA = x[2]
    thetaAcA = x[3]
    AcBA1 = x[4]
    thetaAcBA1 = x[5]
```

```
    AcB1 = x[6]
    thetaAcB1 = x[7]
    AcP1 = x[8]
    thetaAcP1 = x[9]

    x =
Accel(a,b,c,theta2,theta32,theta42,omega2,omega32,omega42,alpha2
,rp,delta3)

    alpha32 = x[0]
    alpha42 = x[1]
    AcBA2 = x[4]
    thetaAcBA2 = x[5]
    AcB2 = x[6]
    thetaAcB2 = x[7]
    AcP2 = x[8]
    thetaAcP2 = x[9]

    if config == 1:
        return
theta31,theta41,omega31,omega41,va,thetaa,vb1,thetab1,vp1,thetap
1,AcA,thetaAcA,AcB1,thetaAcB1,AcP1,thetaAcP1,alpha31,alpha41
    elif config == 2:
        return
theta32,theta42,omega32,omega42,va,thetaa,vb2,thetab2,vp2,thetap
2,AcA,thetaAcA,AcB2,thetaAcB2,AcP2,thetaAcP2,alpha32,alpha42
    else:
        raise Exception('You have inputted an invalid
configuration. The options are "1" for open, "2" for crossed')

x1 = Loop(a,b,c,d,theta2,omega2,alpha2,rp,delta3,1)
x2 = Loop(a,b,c,d,theta2,omega2,alpha2,rp,delta3,2)

#displays the calculated values
print "theta31 = {}\ntheta32 = {}\ntheta41 = {}\ntheta42 =
{}\nomega31 = {}\nomega32 = {}\nomega41 = {}\nomega42 = {}\nvA =
{} at {}\nvB1 = {} at {}\n\
vB2 = {} at {}\nvP1 = {} at {}\nvP2 = {} at {}\nAcA = {} at
{}\nAcB1 = {} at {}\nAcB2 = {} at {}\nAcP1 = {} at {}\nAcP2 = {}
at {}\nalpha31 = {}\n\
alpha32 = {}\nalpha41 = {}\nalpha42 = {}".format(\
```

```
x1[0]*(180/pi),x2[0]*(180/pi),x1[1]*(180/pi),x2[1]*(180/pi),x1[2
],x2[2],x1[3],x2[3],x1[4],x1[5]*(180/pi),x1[6],x1[7]*(180/pi),x2
[6],x2[7]*(180/pi),\

x1[8],x1[9]*(180/pi),x2[8],x2[9]*(180/pi),x1[10],x1[11]*(180/pi)
,x1[12],x1[13]*(180/pi),x2[12],x2[13]*(180/pi),x1[14],x1[15]*(18
0/pi),x2[14],x2[15]*(180/pi),\
    x1[16],x2[16],x1[17],x2[17])

x = Loop(a,b,c,d,theta2,omega2,alpha2,rp,delta3,config)

O2 = vector(0,0,0)
O4 = vector(d, 0, 0)
RA = vector(a*cos(theta2),a*sin(theta2),0)
RB = vector(c*cos(x[1]),c*sin(x[1]), 0) + O4
RPA = vector(rp*cos(x[0]+delta3), rp*sin(x[0]+delta3), 0)
RP = RA + RPA

Link2 = curve(pos = [(O2), (RA)], radius = max(a,b,c,d)/100,
display = scene)
Link3 = curve(pos = [(RA), (RB)], radius = max(a,b,c,d)/100,
display = scene)
Link4 = curve(pos = [(O4), (RB)], radius = max(a,b,c,d)/100,
display = scene)
LinkP = curve(pos = [(RA), (RP)], radius = max(a,b,c,d)/100,
display = scene)
LinkG = curve(pos = [(RP), (RB)], radius = max(a,b,c,d)/100,
display = scene)

theta2s = arange(0, 2*pi + deltatheta2,deltatheta2)
thetafollower = arange(theta2, theta2+2*pi+deltatheta2,
deltatheta2)
aPs = []
vPs = []
for n in theta2s:

aPs.append(Loop(a,b,c,d,n,omega2,alpha2,rp,delta3,config)[14])

vPs.append(Loop(a,b,c,d,n,omega2,alpha2,rp,delta3,config)[8])
```

```
scalefactorap = 5/max(aPs)
scalefactorvp = 5/max(vPs)
print "max(aPs) = ",max(aPs)

scene1 = display(x=900, y=0, width=xdim*(700/1600),
height=ydim*(425/900))
yaxisaP = curve(pos=[(theta2,min(aPs)*scalefactorap,0),
(theta2,max(aPs)*scalefactorap,0)], radius = 1/40, display =
scene1)
xaxisaP = curve(pos=[(theta2,min(aPs)*scalefactorap,0),
(2*pi+theta2,min(aPs)*scalefactorap,0)], radius = 1/40, display
= scene1)
xaxistextaP = text(pos = (pi,max(aPs)*scalefactorap+0.5,0),
align = 'center', text = "Magnitude of Acceleration (Not to
Scale)", height = 0.2, depth = 0.001)

scene2 = display(x=900, y=425, width=xdim*(700/1600),
height=ydim*(425/900))
yaxisvP = curve(pos=[(theta2,min(vPs)*scalefactorvp,0),
(theta2,max(vPs)*scalefactorvp,0)], radius = 1/40, display =
scene2)
xaxisvP = curve(pos=[(theta2,min(vPs)*scalefactorvp,0),
(2*pi+theta2,min(vPs)*scalefactorvp,0)], radius = 1/40, display
= scene2)
xaxistextaP = text(pos = (pi,max(aPs)*scalefactorap+0.5,0),
align = 'center', text = "Magnitude of Velocity (Not to Scale)",
height = 0.2, depth = 0.001)

aPcurve = curve(radius = 1/30, display = scene1)
vPcurve = curve(radius = 1/30, display = scene2)
for n in thetafollower:

aPcurve.append(pos=(n,Loop(a,b,c,d,n,omega2,alpha2,rp,delta3,con
fig)[14]*scalefactorap,0))

vPcurve.append(pos=(n,Loop(a,b,c,d,n,omega2,alpha2,rp,delta3,con
fig)[8]*scalefactorvp,0))

scene1.center =
(pi+theta2,scalefactorap*((max(aPs)+min(aPs))/2),0)
scene1.range = 5
```

```
scene2.center =
(pi+theta2,scalefactorvp*((max(vPs)+min(vPs))/2),0)
scene2.range = 5

followeraP = sphere(pos=(theta2,scalefactorap*x[6],0), radius =
1/8, color = (1,1,1), display = scene1)
followervP = sphere(pos=(theta2,scalefactorap*x[8],0), radius =
1/8, color = (1,1,1), display = scene2)

followercounter = 0

while true:
    rate(100)
    theta2 = theta2 + deltatheta2

    x = Loop(a,b,c,d,theta2,omega2,alpha2,rp,delta3,config)

    RA = vector(a*cos(theta2),a*sin(theta2), 0)
    RB = vector(c*cos(x[1]),c*sin(x[1]), 0) + O4
    RPA = vector(rp*cos(x[0]+delta3), rp*sin(x[0]+delta3), 0)
    RP = RA + RPA

    Link2.pos = [(O2), (RA)]
    Link3.pos = [(RA), (RB)]
    Link4.pos = [(O4), (RB)]
    LinkP.pos = [(RA), (RP)]
    LinkG.pos = [(RP), (RB)]

    followeraP.pos =
(thetafollower[followercounter],scalefactorap*x[14],0)
    followervP.pos =
(thetafollower[followercounter],scalefactorvp*x[8],0)

    followercounter = followercounter + 1

    if followercounter == len(thetafollower):
        followercounter = 0
    else:
        pass
```

Appendix 3

```
from __future__ import division
from visual import *

testsolver = input("Enter 1 for position of 4-bar linkage, 2 for
position of crank-slider, 3 for velocity of \
4-bar linkage, 4 for velocity of crank-slider, 5 for
acceleration of 4-bar linkage, 6 for acceleration \
of crank-slider, 7 for forces on 4-bar linkage, 8 for forces on
crank slider, or 9 for cam and follower \
position, velocity, acceleration, jerk, and vibration: ")

print ""

if testsolver == 1:

    d = 75 #link 1 length
    print "d = {} mm".format(d)
    a = 50 #link 2 length
    print "a = {} mm".format(a)
    b = 75 #link 3 length
    print "b = {} mm".format(b)
    c = 50 #link 4 length
    print "c = {} mm".format(c)
    theta2 = 1
    print "theta2 = {} rad".format(theta2)
    print ""

    #**WORKSPACE**

    #Vector Loop equations substitution values begin here
    k1 = d/a
    k2 = d/c
    k3 = (a**2-b**2+c**2+d**2)/(2*a*c)
    k4 = d/b
    k5 = (c**2-d**2-a**2-b**2)/(2*a*b)

    A = cos(theta2)-k1-k2*cos(theta2)+k3
    B = -2*sin(theta2)
    C = k1-(k2+1)*cos(theta2)+k3
```

```
    D = cos(theta2)-k1+k4*cos(theta2)+k5
    E = -2*sin(theta2)
    F = k1+(k4-1)*cos(theta2)+k5

    #Vector Loop equations substitution values have ended

    n31 = -E-sqrt(E**2-4*D*F) #The open angle theta3 numerator
is calculated
    n32 = -E+sqrt(E**2-4*D*F) #The crossed ange theta3 numerator
is calculated
    d3 = 2*D #The denominator for the theta3 value is caclulated

    n41 = -B-sqrt(B**2-4*A*C) #The open angle theta4 numerator
is calculated
    n42 = -B+sqrt(B**2-4*A*C) #The crossed ange theta4 numerator
is calculated
    d4 = 2*A #The denominator for the theta4 value is caclulated

    #if the denominator is zero, print error messsage
    #if else, calculate values of theta3 (degrees) using Vector
Loop equation
    if d3 == 0:
        disp("you are dividing by zero in your theta3 arctan
function")
    else:
        theta31 = 2*atan(n31/d3)
        theta32 = 2*atan(n32/d3)

    #if the denominator is zero, print error messsage
    #if else, calculate values of theta4 (degrees) using Vector
Loop equation
    if d4 == 0:
        disp("you are dividing by zero in your theta4 arctan
function")
        theta41 = pi
        theta42 = pi
    else:
        theta41 = 2*atan(n41/d4)
        theta42 = 2*atan(n42/d4)

    links = [a,b,c,d]
```

```
    sortlink = sorted(links)
    testGrashof1 = sortlink[0] + sortlink[3]
    testGrashof2 = sortlink[1] + sortlink[2]

    if testGrashof1 <= testGrashof2:

        print "The linkage is Grashof"

    else:

        print "The linkage is non-Grashof"

    #displays the calculated values
    print "theta31 = {} deg".format(theta31*(180/pi))
    print "theta32 = {} deg".format(theta32*(180/pi))
    print "theta41 = {} deg".format(theta41*(180/pi))
    print "theta42 = {} deg".format(theta42*(180/pi))

elif testsolver == 2:

    a = 50 #input("Link 2 (a) = ") #Prompts the user for link 2
length
    print "a = {} mm".format(a)
    b = 75 #input("Link 3 (b) = ") #Prompts the user for link 3
length
    print "b = {} mm".format(b)
    c = 42.07355 #input("Link 4 (c) = ") #Prompts the user for
link 4 length
    print "c = {} mm".format(c)
    theta2 = 1
    print "theta2 = {} rad".format(theta2)
    print ""

    frac = (a*sin(theta2)-c)/b #defines fraction of substitution
values for ease
    #of use in asin function later

    theta31 = asin(-frac) + pi #calculates theta3 open (degrees)
    theta32 = asin(frac) #calculates theta3 crossed (degrees)
    d1 = a*cos(theta2) - b*cos(theta31) #calculates open
distance
```

```
    d2 = a*cos(theta2) - b*cos(theta32) #calculates crossed
distance

    print "theta31 = {} deg".format(theta31*(180/pi))
    print "theta32 = {} deg".format(theta32*(180/pi))
    print "d1 = {} mm".format(d1)
    print "d2 = {} mm".format(d2)

elif testsolver == 3:

    d = 75 #link 1 length
    a = 50 #link 2 length
    b = 75 #link 3 length
    c = 50 #link 4 length
    omega2 = 1 #omega2 (rad/sec)
    theta2 = 1

    print "d = {} mm".format(d)
    print "a = {} mm".format(a)
    print "b = {} mm".format(b)
    print "c = {} mm".format(c)
    print "theta2 = {} rad".format(theta2)
    print "omega2 = {} rad/sec".format(omega2)
    print ""

#**WORKSPACE**

    #Vector Loop equations substituion values begin here
    k1 = d/a
    k2 = d/c
    k3 = (a**2-b**2+c**2+d**2)/(2*a*c)
    k4 = d/b
    k5 = (c**2-d**2-a**2-b**2)/(2*a*b)

    A = cos(theta2)-k1-k2*cos(theta2)+k3
    B = -2*sin(theta2)
    C = k1-(k2+1)*cos(theta2)+k3
    D = cos(theta2)-k1+k4*cos(theta2)+k5
    E = -2*sin(theta2)
    F = k1+(k4-1)*cos(theta2)+k5
```

```
    #Vector Loop equations substitution values have ended

    n31 = -E-sqrt(E**2-4*D*F) #The open angle theta3 numerator
is calculated
    n32 = -E+sqrt(E**2-4*D*F) #The crossed ange theta3 numerator
is calculated
    d3 = 2*D #The denominator for the theta3 value is caclulated

    n41 = -B-sqrt(B**2-4*A*C) #The open angle theta4 numerator
is calculated
    n42 = -B+sqrt(B**2-4*A*C) #The crossed ange theta4 numerator
is calculated
    d4 = 2*A #The denominator for the theta4 value is caclulated

    #if the denominator is zero, print error messsage
    #if else, calculate values of theta3 (degrees) using Vector
Loop equation
    if d3 == 0:
        disp("you are dividing by zero in your theta3 arctan
function")
    else:
        theta31 = 2*atan(n31/d3)
        theta32 = 2*atan(n32/d3)

    #if the denominator is zero, print error messsage
    #if else, calculate values of theta4 (degrees) using Vector
Loop equation
    if d4 == 0:
        disp("you are dividing by zero in your theta4 arctan
function")
        theta41 = pi
        theta42 = pi
    else:
        theta41 = 2*atan(n41/d4)
        theta42 = 2*atan(n42/d4)

    #calculates values for open and crossed omega 3 (rad/time)
    omega31 = (a*omega2*sin(theta41-theta2))/(b*sin(theta31-
theta41))
    omega32 = (a*omega2*sin(theta42-theta2))/(b*sin(theta32-
theta42))
```

```
    #calculates values for open and crossed omega 4 (rad/time)
    omega41 = (a*omega2*sin(theta2-theta31))/(c*sin(theta41-
theta31))
    omega42 = (a*omega2*sin(theta2-theta32))/(c*sin(theta42-
theta32))

    #calculates value of the magnitude of va (length/time)
    vax = -a*omega2*sin(theta2)
    vay = a*omega2*cos(theta2)
    va = sqrt(vax**2+vay**2)

    thetaa = atan2(vay,vax)

    #calculates value of the magnitude of vb1 (length/time)
    vbx1 = -c*omega41*sin(theta41)
    vby1 = c*omega41*cos(theta41)
    vb1 = sqrt(vbx1**2+vby1**2)

    #calculates value of the direction of vb1 (deg)
    thetab1 = atan2(vby1,vbx1)

    #calculates value of the magnitude of vb2 (length/time)
    vbx2 = -c*omega42*sin(theta42)
    vby2 = c*omega42*cos(theta42)
    vb2 = sqrt(vbx2**2+vby2**2)

    #calculates value of the direction of vb1 (deg)
    thetab2 = atan2(vby2,vbx2)

    print "theta31 = {} deg".format(theta31*(180/pi))
    print "theta32 = {} deg".format(theta32*(180/pi))
    print "theta41 = {} deg".format(theta41*(180/pi))
    print "theta42 = {} deg".format(theta42*(180/pi))
    print "omega31 = {} rad/sec".format(omega31)
    print "omega32 = {} rad/sec".format(omega32)
    print "omega41 = {} rad/sec".format(omega41)
    print "omega42 = {} rad/sec".format(omega42)
    print "vA = {} mm/sec at {} deg".format(va,thetaa*(180/pi))
    print "vB1 = {} mm/sec at {}
deg".format(vb1,thetab1*(180/pi))
```

```
    print "vB2 = {} mm/sec at {}
deg".format(vb2,thetab2*(180/pi))

elif testsolver == 4:

    a = 50 #link 2 length
    b = 75 #link 3 length
    c = 42.07355 #offset length
    #theta2deg =  #theta2 (degrees)
    omega2 = 1 #omega2 (rad/time)
    theta2 = 1

    print "a = {} mm".format(a)
    print "b = {} mm".format(b)
    print "c = {} mm".format(c)
    print "theta2 = {} rad".format(theta2)
    print "omega2 = {} rad/sec".format(omega2)
    print ""

    frac = (a*sin(theta2)-c)/b #defines fraction of substitution
values for ease
    #of use in asin function later

    theta31 = asin(-frac) + pi #calculates theta3 open (degrees)
    theta32 = asin(frac) #calculates theta3 crossed (degrees)
    d1 = a*cos(theta2) - b*cos(theta31) #calculates open
distance
    d2 = a*cos(theta2) - b*cos(theta32) #calculates crossed
distance

    #calculates omega 3, open and crossed
    omega31 = (a*cos(theta2)*omega2)/(b*cos(theta31))
    omega32 = (a*cos(theta2)*omega2)/(b*cos(theta32))

    #calculates value of the magnitude of va (length/time)
    vax = -a*omega2*sin(theta2)
    vay = a*omega2*cos(theta2)
    va = sqrt(vax**2+vay**2)

    #calculates the value of theta vA
    thetaa = atan2(vay,vax)
```

```
    #calculates vb, open and crossed
    vb1 = -a*omega2*sin(theta2)+b*omega31*sin(theta31)
    vb2 = -a*omega2*sin(theta2)+b*omega32*sin(theta32)

    print "theta31 = {} deg".format(theta31*(180/pi))
    print "theta32 = {} deg".format(theta32*(180/pi))
    print "d1 = {} mm".format(d1)
    print "d2 = {} mm".format(d2)
    print "omega31 = {} rad/sec".format(omega31)
    print "omega32 = {} rad/sec".format(omega32)
    print "vA = {} mm/sec at {} deg".format(va,thetaa*(180/pi))
    print "vB1 = {} mm/sec".format(vb1)
    print "vB2 = {} mm/sec".format(vb2)

elif testsolver == 5:

    d = 75 #link 1 length
    a = 50 #link 2 length
    b = 75 #link 3 length
    c = 50 #link 4 length
    #theta2deg =  #theta2 (degrees)
    omega2 = 1 #omega2 (rad/sec)
    alpha2 = 1 #alpha2 (rad/sec**2)
    theta2 = 1

    print "d = {} mm".format(d)
    print "a = {} mm".format(a)
    print "b = {} mm".format(b)
    print "c = {} mm".format(c)
    print "theta2 = {} rad".format(theta2)
    print "omega2 = {} rad/sec".format(omega2)
    print "alpha2 = {} rad/sec/sec".format(alpha2)
    print ""

    config = 0

    def Loop(a,b,c,d,theta2,omega2,alpha2,config):

        #Vector Loop equations substituion values begin here
        k1 = d/a
```

```
        k2 = d/c
        k3 = (a**2-b**2+c**2+d**2)/(2*a*c)
        k4 = d/b
        k5 = (c**2-d**2-a**2-b**2)/(2*a*b)

        A = cos(theta2)-k1-k2*cos(theta2)+k3
        B = -2*sin(theta2)
        C = k1-(k2+1)*cos(theta2)+k3
        D = cos(theta2)-k1+k4*cos(theta2)+k5
        E = -2*sin(theta2)
        F = k1+(k4-1)*cos(theta2)+k5

        #Vector Loop equations substitution values have ended

        n31 = -E-sqrt(E**2-4*D*F) #The open angle theta3
numerator is calculated
        n32 = -E+sqrt(E**2-4*D*F) #The crossed ange theta3
numerator is calculated
        d3 = 2*D #The denominator for the theta3 value is
caclulated

        n41 = -B-sqrt(B**2-4*A*C) #The open angle theta4
numerator is calculated
        n42 = -B+sqrt(B**2-4*A*C) #The crossed ange theta4
numerator is calculated
        d4 = 2*A #The denominator for the theta4 value is
caclulated

        #if the denominator is zero, print error messsage
        #if else, calculate values of theta3 (degrees) using
Vector Loop equation
        if d3 == 0:
            disp("you are dividing by zero in your theta3 arctan
function")
        else:
            theta31 = 2*atan(n31/d3)
            theta32 = 2*atan(n32/d3)

        #if the denominator is zero, print error messsage
        #if else, calculate values of theta4 (degrees) using
Vector Loop equation
```

```
        if d4 == 0:
            disp("you are dividing by zero in your theta4 arctan
function")
            theta41 = pi
            theta42 = pi
        else:
            theta41 = 2*atan(n41/d4)
            theta42 = 2*atan(n42/d4)

        #calculates values for open and crossed omega 3
(rad/time)
        omega31 = (a*omega2*sin(theta41-theta2))/(b*sin(theta31-
theta41))
        omega32 = (a*omega2*sin(theta42-theta2))/(b*sin(theta32-
theta42))

        #calculates values for open and crossed omega 4
(rad/time)
        omega41 = (a*omega2*sin(theta2-theta31))/(c*sin(theta41-
theta31))
        omega42 = (a*omega2*sin(theta2-theta32))/(c*sin(theta42-
theta32))

        #calculates value of the magnitude of va (length/time)
        vax = -a*omega2*sin(theta2)
        vay = a*omega2*cos(theta2)
        va = sqrt(vax**2+vay**2)

        thetaa = atan2(vay,vax)

        #calculates value of the magnitude of vb1 (length/time)
        vbx1 = -c*omega41*sin(theta41)
        vby1 = c*omega41*cos(theta41)
        vb1 = sqrt(vbx1**2+vby1**2)

        #calculates value of the direction of vb1 (deg)
        thetab1 = atan2(vby1,vbx1)

        #calculates value of the magnitude of vb2 (length/time)
        vbx2 = -c*omega42*sin(theta42)
        vby2 = c*omega42*cos(theta42)
```

```
        vb2 = sqrt(vbx2**2+vby2**2)

        #calculates value of the direction of vb1 (deg)
        thetab2 = atan2(vby2,vbx2)

        def
Accel(a,b,c,theta2,theta3,theta4,omega2,omega3,omega4,alpha2):

            A = c*sin(theta4)
            B = b*sin(theta3)
            C =
a*alpha2*sin(theta2)+a*omega2**2*cos(theta2)+b*omega3**2*cos(the
ta3)-c*omega4**2*cos(theta4)
            D = c*cos(theta4)
            E = b*cos(theta3)
            F = a*alpha2*cos(theta2)-a*omega2**2*sin(theta2)-
b*omega3**2*sin(theta3)+c*omega4**2*sin(theta4)

            alpha3 = (C*D-A*F)/(A*E-B*D)
            alpha4 = (C*E-B*F)/(A*E-B*D)

            AcAx = -a*alpha2*sin(theta2)-a*omega2**2*cos(theta2)
            AcAy = a*alpha2*cos(theta2)-a*omega2**2*sin(theta2)

            AcA = sqrt(AcAx**2+AcAy**2)
            thetaAcA = atan2(AcAy,AcAx)

            AcBAx = -b*alpha3*sin(theta3)-
b*omega3**2*cos(theta3)
            AcBAy = b*alpha3*cos(theta3)-b*omega3**2*sin(theta3)

            AcBA = sqrt(AcBAx**2+AcBAy**2)
            thetaAcBA = atan2(AcBAy,AcBAx)

            AcBx = -c*alpha4*sin(theta4)-c*omega4**2*cos(theta4)
            AcBy = c*alpha4*cos(theta4)-c*omega4**2*sin(theta4)

            AcB = sqrt(AcBx**2+AcBy**2)
            thetaAcB = atan2(AcBy,AcBx)
```

```
            return
alpha3,alpha4,AcA,thetaAcA,AcBA,thetaAcBA,AcB,thetaAcB

        x =
Accel(a,b,c,theta2,theta31,theta41,omega2,omega31,omega41,alpha2
)

        alpha31 = x[0]
        alpha41 = x[1]
        AcA = x[2]
        thetaAcA = x[3]
        AcBA1 = x[4]
        thetaAcBA1 = x[5]
        AcB1 = x[6]
        thetaAcB1 = x[7]

        x =
Accel(a,b,c,theta2,theta32,theta42,omega2,omega32,omega42,alpha2
)

        alpha32 = x[0]
        alpha42 = x[1]
        AcBA2 = x[4]
        thetaAcBA2 = x[5]
        AcB2 = x[6]
        thetaAcB2 = x[7]

        if config == 1:
            return
theta31,theta41,omega31,omega41,va,thetaa,vb1,thetab1,AcA,thetaA
cA,AcB1,thetaAcB1,alpha31,alpha41
        elif config == 2:
            return
theta32,theta42,omega32,omega42,va,thetaa,vb2,thetab2,AcA,thetaA
cA,AcB2,thetaAcB2,alpha32,alpha42
        else:
            raise Exception('You have inputted an invalid
configuration. The options are "1" for open, "2" for crossed')

    x1 = Loop(a,b,c,d,theta2,omega2,alpha2,1)
    x2 = Loop(a,b,c,d,theta2,omega2,alpha2,2)
```

```
    #displays the calculated values
    print "theta31 = {} deg\ntheta32 = {} deg\ntheta41 = {}
deg\ntheta42 = {} deg\nomega31 = {} rad/sec\nomega32 = {}
rad/sec\n\
    omega41 = {} rad/sec\nomega42 = {} rad/sec\nvA = {} mm/sec
at {} deg\nvB1 = {} mm/sec at {} deg\n\
    vB2 = {} mm/sec at {} deg\nAcA = {} mm/sec/sec at {}
deg\nAcB1 = {} mm/sec/sec at {} deg\n\
    AcB2 = {} mm/sec/sec at {} deg\nalpha31 = {} rad/sec/sec\n\
    alpha32 = {} rad/sec/sec\nalpha41 = {} rad/sec/sec\nalpha42
= {} rad/sec/sec".format(\

x1[0]*(180/pi),x2[0]*(180/pi),x1[1]*(180/pi),x2[1]*(180/pi),x1[2
],x2[2],x1[3],x2[3],x1[4],x1[5]*(180/pi),x1[6],x1[7]*(180/pi),x2
[6],x2[7]*(180/pi),\

x1[8],x1[9]*(180/pi),x1[10],x1[11]*(180/pi),x2[10],x2[11]*(180/p
i),\
        x1[12],x2[12],x1[13],x2[13])


elif testsolver == 6:

    a = 50 #input("Link 2 (a) = ") #link 2 length
    b = 75 #input("Link 3 (b) = ") #link 3 length
    c = 42.07355 #input("Offset = ") #offset length
    #theta2deg =  #theta2 (degrees)
    omega2 = 1 #omega2 (rad/time)
    alpha2 = 1 #alpha2 (rad/sec**2)
    theta2 = 1

    print "a = {} mm".format(a)
    print "b = {} mm".format(b)
    print "c = {} mm".format(c)
    print "theta2 = {} rad".format(theta2)
    print "omega2 = {} rad/sec".format(omega2)
    print "alpha2 = {} rad/sec/sec".format(alpha2)
    print ""

    config = 1
```

```
    def Loop(a,b,c,theta2,omega2,alpha2,config):

        frac = (a*sin(theta2)-c)/b #defines fraction of
substitution values for ease
        #of use in asin function later

        theta31 = asin(-frac) + pi #calculates theta3 open
(degrees)
        theta32 = asin(frac) #calculates theta3 crossed
(degrees)
        d1 = a*cos(theta2) - b*cos(theta31) #calculates open
distance
        d2 = a*cos(theta2) - b*cos(theta32) #calculates crossed
distance

        #calculates omega 3, open and crossed
        omega31 = (a*cos(theta2)*omega2)/(b*cos(theta31))
        omega32 = (a*cos(theta2)*omega2)/(b*cos(theta32))

        #calculates value of the magnitude of va (length/time)
        vax = -a*omega2*sin(theta2)
        vay = a*omega2*cos(theta2)
        va = sqrt(vax**2+vay**2)

        #calculates the value of theta vA
        thetaa = atan2(vay,vax)

        #calculates vb, open and crossed
        vb1 = -a*omega2*sin(theta2)+b*omega31*sin(theta31)
        vb2 = -a*omega2*sin(theta2)+b*omega32*sin(theta32)

        AcAx = -a*alpha2*sin(theta2)-a*omega2**2*cos(theta2)
        AcAy = a*alpha2*cos(theta2)-a*omega2**2*sin(theta2)
        AcA = sqrt(AcAx**2+AcAy**2)
        thetaAcA = atan2(AcAy,AcAx)

        alpha31 = (a*alpha2*cos(theta2)-
a*omega2**2*sin(theta2)+b*omega31**2*sin(theta31))/(b*cos(theta3
1))
```

```
        alpha32 = (a*alpha2*cos(theta2)-
a*omega2**2*sin(theta2)+b*omega32**2*sin(theta32))/(b*cos(theta3
2))

        Ad1 = -a*alpha2*sin(theta2)-
a*omega2**2*cos(theta2)+b*alpha31*sin(theta31)+b*omega31**2*cos(
theta31)
        Ad2 = -a*alpha2*sin(theta2)-
a*omega2**2*cos(theta2)+b*alpha32*sin(theta32)+b*omega31**2*cos(
theta32)

        if config == 1:
            return
theta31,d1,va,thetaa,omega31,vb1,AcA,thetaAcA,alpha31,Ad1
        elif config == 2:
            return
theta32,d2,va,thetaa,omega32,vb2,AcA,thetaAcA,alpha32,Ad2
        else:
            raise Exception('You have inputted an invalid
configuration. The options are "1" for open, "2" for crossed')

    #displays calculated values of theta3 (degrees) and distance

    x1 = Loop(a,b,c,theta2,omega2,alpha2,1)
    x2 = Loop(a,b,c,theta2,omega2,alpha2,2)

    print "theta31 = {} deg\ntheta32 = {} deg\nd1 = {} mm\nd2 =
{} mm\nva = {} mm/sec at {} deg\nomega31 = {} rad/sec\nomega32 =
{} rad/sec\n\
    vb1 = {} mm/sec\nvd2 = {}m mm/sec\nAcA = {} mm/sec/sec at {}
deg\nalpha31 = {} rad/sec/sec\n\
    alpha32 = {} rad/sec/sec\nAd1 = {} mm/sec/sec\nAd2 = {}
mm/sec/sec\n".format(x1[0]*(180/pi),x2[0]*(180/pi),x1[1],x2[1],x
1[2],x1[3]\

*(180/pi),x1[4],x2[4],x1[5],x2[5],x1[6],x1[7]*(180/pi),x1[8],x2[
8],x1[9],x2[9])


elif testsolver == 7:
```

```
d = 0.075 #link 1 length
a = 0.050 #link 2 length
b = 0.075 #link 3 length
c = 0.050 #link 4 length
omega2 = 1 #omega2 (rad/sec)
alpha2 = 1 #alpha2 (rad/sec**2)
config = 1 # open (1) or crossed (2) configuration
theta2 = 1 #theta2deg*(pi/180)

widthAl = 0.012 #m
thicknessAl = 0.007 #m
widthTi = 0.015 #m
thicknessTi = 0.005 #m

rhoAl = 2810 #kg/m**3
rhoTi = 4429 #kg/m**3

print "d = {} m".format(d)
print "a = {} m".format(a)
print "b = {} m".format(b)
print "c = {} m".format(c)
print "theta2 = {} rad".format(theta2)
print "omega2 = {} rad/sec".format(omega2)
print "alpha2 = {} rad/sec/sec".format(alpha2)
print "widthAl = {} m".format(widthAl)
print "thicknessAl = {} m".format(thicknessAl)
print "widthTi = {} m".format(widthTi)
print "thicknessTi = {} m".format(widthTi)
print "rhoAl = {} kg/m^3".format(rhoAl)
print "rhoTi = {} kg/m^3".format(rhoTi)

#Vector Loop equations substituion values begin here
k1 = d/a
k2 = d/c
k3 = (a**2-b**2+c**2+d**2)/(2*a*c)
k4 = d/b
k5 = (c**2-d**2-a**2-b**2)/(2*a*b)

A = cos(theta2)-k1-k2*cos(theta2)+k3
B = -2*sin(theta2)
C = k1-(k2+1)*cos(theta2)+k3
```

```
    D = cos(theta2)-k1+k4*cos(theta2)+k5
    E = -2*sin(theta2)
    F = k1+(k4-1)*cos(theta2)+k5

    #Vector Loop equations substitution values have ended

    n31 = -E-sqrt(E**2-4*D*F) #The open angle theta3 numerator
is calculated
    n32 = -E+sqrt(E**2-4*D*F) #The crossed ange theta3 numerator
is calculated
    d3 = 2*D #The denominator for the theta3 value is caclulated

    n41 = -B-sqrt(B**2-4*A*C) #The open angle theta4 numerator
is calculated
    n42 = -B+sqrt(B**2-4*A*C) #The crossed ange theta4 numerator
is calculated
    d4 = 2*A #The denominator for the theta4 value is caclulated

    #if the denominator is zero, print error messsage
    #if else, calculate values of theta3 (degrees) using Vector
Loop equation
    if d3 == 0:
        disp("you are dividing by zero in your theta3 arctan
function")
    else:
        theta31 = 2*atan(n31/d3)
        theta32 = 2*atan(n32/d3)

    #if the denominator is zero, print error messsage
    #if else, calculate values of theta4 (degrees) using Vector
Loop equation
    if d4 == 0:
        disp("you are dividing by zero in your theta4 arctan
function")
        theta41 = pi
        theta42 = pi
    else:
        theta41 = 2*atan(n41/d4)
        theta42 = 2*atan(n42/d4)

    #calculates values for open and crossed omega 3 (rad/time)
```

```
    omega31 = (a*omega2*sin(theta41-theta2))/(b*sin(theta31-
theta41))
    omega32 = (a*omega2*sin(theta42-theta2))/(b*sin(theta32-
theta42))

    #calculates values for open and crossed omega 4 (rad/time)
    omega41 = (a*omega2*sin(theta2-theta31))/(c*sin(theta41-
theta31))
    omega42 = (a*omega2*sin(theta2-theta32))/(c*sin(theta42-
theta32))

    A1 = c*sin(theta41)
    B1 = b*sin(theta31)
    C1 =
a*alpha2*sin(theta2)+a*omega2**2*cos(theta2)+b*omega31**2*cos(th
eta31)-c*omega41**2*cos(theta41)
    D1 = c*cos(theta41)
    E1 = b*cos(theta31)
    F1 = a*alpha2*cos(theta2)-a*omega2**2*sin(theta2)-
b*omega31**2*sin(theta31)+c*omega41**2*sin(theta41)

    alpha31 = (C1*D1-A1*F1)/(A1*E1-B1*D1)
    alpha41 = (C1*E1-B1*F1)/(A1*E1-B1*D1)

    A2 = c*sin(theta42)
    B2 = b*sin(theta32)
    C2 =
a*alpha2*sin(theta2)+a*omega2**2*cos(theta2)+b*omega32**2*cos(th
eta32)-c*omega42**2*cos(theta42)
    D2 = c*cos(theta42)
    E2 = b*cos(theta32)
    F2 = a*alpha2*cos(theta2)-a*omega2**2*sin(theta2)-
b*omega32**2*sin(theta32)+c*omega42**2*sin(theta42)

    alpha32 = (C2*D2-A2*F2)/(A2*E2-B2*D2)
    alpha42 = (C2*E2-B2*F2)/(A2*E2-B2*D2)

    AcAx = -a*alpha2*sin(theta2)-a*omega2**2*cos(theta2)
    AcAy = a*alpha2*cos(theta2)-a*omega2**2*sin(theta2)

    aA = vector(AcAx,AcAy,0)
```

```
    O2 = vector(0,0,0)
    O4 = vector(d,0,0)

    A = vector(cos(theta2)*a,sin(theta2)*a,0)
    G2 = 0.5*A

    RB1 = vector(cos(theta41)*c,sin(theta41)*c,0)
    RG41 = 0.5*RB1
    B1 = RB1 + O4
    G41 = RG41 + O4

    RG31 = 0.5*(B1 - A)
    G31 = RG31 + A

    R12 = -G2
    R32 = G2

    R231 = -RG31
    R431 = RG31

    R141 = -RG41
    R341 = RG41

    aG2 = cross(vector(0,0,alpha2),G2)-omega2**2*G2

    aG31 = cross(vector(0,0,alpha31),RG31)-omega31**2*RG31 + aA

    aG41 = cross(vector(0,0,alpha41),RG41)-omega41**2*RG41

    m2 = widthAl*thicknessAl*a*(rhoAl) #kg
    m3 = widthTi*thicknessTi*b*(rhoTi) #kg
    m4 = widthAl*thicknessAl*c*(rhoAl) #kg

    I2 = (m2*(widthAl**2 + a**2))/12
    I3 = (m3*(widthTi**2 + b**2))/12
    I4 = (m4*(widthAl**2 + c**2))/12

    MatrixA = matrix([[1,0,1,0,0,0,0,0,0],[0,1,0,1,0,0,0,0,0],[-
R12[1],R12[0],-R32[1],R32[0],0,0,0,0,1],[0,0,-
1,0,1,0,0,0,0],[0,0,0,-1,0,1,0,0,0],\
```

```
            [0,0,R231[1],-R231[0],-
R431[1],R431[0],0,0,0],[0,0,0,0,-1,0,1,0,0],[0,0,0,0,0,-
1,0,1,0],[0,0,0,0,R341[1],-R341[0],-R141[1],R141[0],0]])

    MatrixB =
matrix([[m2*aG2[0]],[m2*aG2[1]],[I2*alpha2],[m3*aG31[0]],[m3*aG3
1[1]],[I3*alpha31],[m4*aG41[0]],[m4*aG41[1]],[I4*alpha41]])

    SOL = MatrixA.getI()*MatrixB

    print ""
    print "O2 = {} m".format(O2)
    print "O4 = {} m".format(O4)
    print "A = {} m".format(A)
    print "G2 = {} m ".format(G2)
    print "RB1 = {} m".format(RB1)
    print "RG41 = {} m".format(RG41)
    print "RG31 = {} m".format(RG31)
    print "G41 = {} m".format(G41)
    print "G31 = {} m".format(G31)
    print "aG2 = {} m/sec/sec".format(aG2)
    print "aG3 = {} m/sec/sec".format(aG31)
    print "aG41 = {} m/sec/sec".format(aG41)
    print "m2 = {} kg".format(m2)
    print "m3 = {} kg".format(m3)
    print "m4 = {} kg".format(m4)
    print "I2 = {} kg m^2".format(I2)
    print "I3 = {} kg m^2".format(I3)
    print "I4 = {} kg m^2".format(I4)
    print "R12 = {} m".format(R12)
    print "R32 = {} m".format(R32)
    print "R231 = {} m".format(R231)
    print "R431 = {} m".format(R431)
    print "R141 = {} m".format(R141)
    print "R341 = {} m".format(R341)
    print



    print
    print "theta31 = {} deg".format(theta31*(180/pi))
    print "theta32 = {} deg".format(theta32*(180/pi))
```

```
    print "theta41 = {} deg".format(theta41*(180/pi))
    print "theta42 = {} deg".format(theta42*(180/pi))
    print "omega31 = {} rad/sec".format(omega31)
    print "omega32 = {} rad/sec".format(omega32)
    print "omega41 = {} rad/sec".format(omega41)
    print "omega42 = {} rad/sec".format(omega42)
    print "AcAx = {} m/sec/sec".format(AcAx)
    print "AcAy = {} m/sec/sec".format(AcAy)
    print "alpha31 = {} rad/sec/sec".format(alpha31)
    print "alpha32 = {} rad/sec/sec".format(alpha32)
    print "alpha41 = {} rad/sec/sec".format(alpha41)
    print "alpha42 = {} rad/sec/sec".format(alpha42)

    print
    print "F12x = {} N".format(SOL[0])
    print "F12y = {} N".format(SOL[1])
    print "F14x = {} N".format(SOL[6])
    print "F14y = {} N".format(SOL[7])
    print "F32x = {} N".format(SOL[2])
    print "F32y = {} N".format(SOL[3])
    print "F43x = {} N".format(SOL[4])
    print "F43y = {} N".format(SOL[5])
    print "T12 = {} N m".format(SOL[8])


elif testsolver == 8:

    a = 0.050 #link 2 length
    b = 0.075 #link 3 length
    c = 0.04207355 #link 4 length
    #theta2deg =  #theta2 (degrees)
    omega2 = 1 #omega2 (rad/sec)
    alpha2 = 1 #alpha2 (rad/sec**2)
    config = 1 #open (1) or crossed (2) configuration
    theta2 = 1 #theta2deg*(pi/180)

    widthAl = 0.012 #m
    thicknessAl = 0.007 #m
    widthTi = 0.015 #m
    thicknessTi = 0.005 #m
```

```
    rhoAl = 2810 #kg/m**3
    rhoTi = 4429 #kg/m**3

    print "a = {} m".format(a)
    print "b = {} m".format(b)
    print "c = {} m".format(c)
    print "theta2 = {} rad".format(theta2)
    print "omega2 = {} rad/sec".format(omega2)
    print "alpha2 = {} rad/sec/sec".format(alpha2)
    print "widthAl = {} m".format(widthAl)
    print "thicknessAl = {} m".format(thicknessAl)
    print "widthTi = {} m".format(widthTi)
    print "thicknessTi = {} m".format(widthTi)
    print "rhoAl = {} kg/m^3".format(rhoAl)
    print "rhoTi = {} kg/m^3".format(rhoTi)

    #Vector Loop equations substituion values begin here

    frac = (a*sin(theta2)-c)/b #defines fraction of substitution
values for ease
    #of use in asin function later

    theta31 = asin(-frac) + pi #calculates theta3 open (degrees)
    theta32 = asin(frac) #calculates theta3 crossed (degrees)
    d1 = a*cos(theta2) - b*cos(theta31) #calculates open
distance
    d2 = a*cos(theta2) - b*cos(theta32) #calculates crossed
distance

    #Vector Loop equations substitution values have ended

    #calculates values for open and crossed omega 3 (rad/time)
    omega31 = (a*cos(theta2)*omega2)/(b*cos(theta31))
    omega32 = (a*cos(theta2)*omega2)/(b*cos(theta32))

    vb1 = -a*omega2*sin(theta2)+b*omega31*sin(theta31)
    vb2 = -a*omega2*sin(theta2)+b*omega32*sin(theta32)

    AcAx = -a*alpha2*sin(theta2)-a*omega2**2*cos(theta2)
    AcAy = a*alpha2*cos(theta2)-a*omega2**2*sin(theta2)
    aA = vector(AcAx,AcAy,0)
```

```
    alpha31 = (a*alpha2*cos(theta2)-
a*omega2**2*sin(theta2)+b*omega31**2*sin(theta31))/(b*cos(theta3
1))
    alpha32 = (a*alpha2*cos(theta2)-
a*omega2**2*sin(theta2)+b*omega32**2*sin(theta32))/(b*cos(theta3
2))

    Ad1 = -a*alpha2*sin(theta2)-
a*omega2**2*cos(theta2)+b*alpha31*sin(theta31)+b*omega31**2*cos(
theta31)
    Ad2 = -a*alpha2*sin(theta2)-
a*omega2**2*cos(theta2)+b*alpha32*sin(theta32)+b*omega31**2*cos(
theta32)

    O2 = vector(0,0,0)
    O4 = vector(d1,0,0)

    A = vector(cos(theta2)*a,sin(theta2)*a,0)
    G2 = 0.5*A

    RB1 = vector(0,c,0)
    B1 = RB1 + O4

    RG31 = 0.5*(B1 - A)
    G31 = RG31 + A

    R12 = -G2
    R32 = G2

    R231 = -RG31
    R131 = RG31

    aG2 = cross(vector(0,0,alpha2),G2)-omega2**2*G2

    aG31 = cross(vector(0,0,alpha31),RG31)-omega31**2*RG31 + aA

    m2 = widthAl*thicknessAl*a*(rhoAl) #kg
    m3 = widthTi*thicknessTi*b*(rhoTi) #kg

    I2 = (m2*(widthAl**2 + a**2))/12
```

```
    I3 = (m3*(widthTi**2 + b**2))/12

    MatrixA = matrix([[1,0,1,0,0,0],[0,1,0,1,0,0],[-
R12[1],R12[0],-R32[1],R32[0],0,1],[0,0,-1,0,0,0],[0,0,0,-
1,1,0],[0,0,R231[1],-R231[0],R131[0],0]])

    MatrixB =
matrix([[m2*aG2[0]],[m2*aG2[1]],[I2*alpha2],[m3*aG31[0]],[m3*aG3
1[1]],[I3*alpha31]])

    SOL = MatrixA.getI()*MatrixB

    print ""
    print "O2 = {} m".format(O2)
    print "A = {} m".format(A)
    print "G2 = {} m".format(G2)
    print "RG31 = {} m".format(RG31)
    print "G31 = {} m".format(G31)
    print "aG2 = {} m/sec/sec".format(aG2)
    print "aG3 = {} m/sec/sec".format(aG31)
    print "m2 = {} kg".format(m2)
    print "m3 = {} kg".format(m3)
    print "I2 = {} kg m^2".format(I2)
    print "I3 = {} kg m^2".format(I3)
    print "R12 = {} m".format(R12)
    print "R32 = {} m".format(R32)
    print "R231 = {} m".format(R231)
    print "R131 = {} m".format(R131)
    print

    print
    print "theta31 = {} deg".format(theta31*(180/pi))
    print "theta32 = {} deg".format(theta32*(180/pi))
    print "d1 = {} m".format(d1)
    print "d2 = {} m".format(d2)
    print "omega31 = {} rad/sec".format(omega31)
    print "omega32 = {} rad/sec".format(omega32)
    print "vB1 = {} m/sec".format(vb1)
    print "vB2 = {} m/sec".format(vb2)
    print "AcB1 = {} m/sec/sec".format(Ad1)
    print "AcB2 = {} m/sec/sec".format(Ad2)
```

```
    print "AcAx = {} m/sec/sec".format(AcAx)
    print "AcAy = {} m/sec/sec".format(AcAy)
    print "alpha31 = {} rad/sec/sec".format(alpha31)
    print "alpha32 = {} rad/sec/sec".format(alpha32)

    print
    print "F12x = {} N".format(SOL[0])
    print "F12y = {} N".format(SOL[1])
    print "F32x = {} N".format(SOL[2])
    print "F32y = {} N".format(SOL[3])
    print "F13x = {} N".format(SOL[4])
    print "T12 = {} N m".format(SOL[5])


elif testsolver == 9:

    from visual.graph import *

    deltat = 0.01

    base = 0.087
    zones = 4

    #Hard-coded inputs:
    h = [0.025,0,-0.025,0]
    t = [0.75,1.25,0.5,0.25]
    func = [1,0,-1,0]
    follower = 0.012
    rotate = 1 #assume CCW rotation
    epsilon = 0
    zeta = 0.045
    mass = 2
    k = 35*(1000/1)  #N/mm to N/m
    preload = 5 #N

    littledelta = preload/k

    print "Base circle = {} m".format(base)
    print "Number of zones = ",zones
    print "The follower rises for each zone are (m): ",h
    print "The times for each zone are (sec): ",t
```

```
    print "The function type for each zone is ('1' for simple
harmonic function, '-1' for cycloidal function, '0' for dwell):
",func
    print "The follower radius is {} m".format(follower)
    print "The rotation is (1 = CCW, -1 = CW): ",rotate
    print "The cam ecentricity is {} m".format(epsilon)
    print "The damping ratio is: ",zeta
    print "The follower mass is {} kg".format(mass)
    print "The spring constant is {} N/m".format(k)
    print "Preload force = {} N".format(preload)
    print "Preload displacement = {} m".format(littledelta)
    print ""

    scene = display (x=0, y=0, width=800, height=800, background
= (1,1,1), foreground = (0,0,0))

    graph1 = gdisplay(x=0, y=0, width=800, height=800,
xtitle="Time (sec)", ytitle="Displacement (m)", background =
(1,1,1), foreground = (0,0,0))
    graph2 = gdisplay(x=0, y=0, width=800, height=800,
xtitle="Time (sec)", ytitle="Velocity (m/sec)", background =
(1,1,1), foreground = (0,0,0))
    graph3 = gdisplay(x=0, y=0, width=800, height=800,
xtitle="Time (sec)", ytitle="Accleration (m/sec/sec)",
background = (1,1,1), foreground = (0,0,0))
    graph4 = gdisplay(x=0, y=0, width=800, height=800,
xtitle="Time (sec)", ytitle="Jerk (m/sec/sec/sec)", background =
(1,1,1), foreground = (0,0,0))
    graph5 = gdisplay(x=0, y=0, width=800, height=800,
xtitle="Time (sec)", ytitle="Pressure Angle (deg)", background =
(1,1,1), foreground = (0,0,0))
    graph6 = gdisplay(x=0, y=0, width=800, height=800,
xtitle="Time (sec)", ytitle="Force on cam (N)", background =
(1,1,1), foreground = (0,0,0))
    graph7 = gdisplay(x=0, y=0, width=800, height=800,
xtitle="Time (sec)", ytitle="Camshaft Torque (N m)", background
= (1,1,1), foreground = (0,0,0))
    graph8 = gdisplay(x=0, y=0, width=800, height=800,
xtitle="Time (sec)", ytitle="Power lost (Due to Camshaft Torque)
((N m)/sec)", background = (1,1,1), foreground = (0,0,0))
```

```
    graph9 = gdisplay(x=0, y=0, width=800, height=800,
xtitle="Time (sec)", ytitle="Power lost (Due to Follower Force)
((N m)/sec)", background = (1,1,1), foreground = (0,0,0))

    sgraph = gcurve(gdisplay = graph1)
    vgraph = gcurve(gdisplay = graph2)
    agraph = gcurve(gdisplay = graph3)
    jgraph = gcurve(gdisplay = graph4)
    pgraph = gcurve(gdisplay = graph5)
    Fcgraph = gcurve(gdisplay = graph6)
    Tcgraph = gcurve(gdisplay = graph7)
    Pgraph = gcurve(gdisplay = graph8)
    Pagraph = gcurve(gdisplay = graph9)

    omega = (360)/sum(t) #deg/sec

    time = arange(0,sum(t),deltat)

    beta = []

    for n in arange(0,zones,1):
        beta.append((t[n]*360)/sum(t))

    deltatheta = deltat*omega

    Rprime = base + follower

    omegan = sqrt(k/mass)

    theta = []
    s = []
    v = []
    a = []
    j = []
    phi = []
    Fc = []
    Tc = []
    P = []
    Pa = []

    counter = 0
```

```
    for n in arange(0,zones,1):

        if func[n] == 1:

            for m in arange(0,t[n],deltat):

                theta.append(counter*deltatheta)

                if n == 0:
                    s.append((h[n]/2)*(1-
cos((pi*m*omega)/beta[n])) + base)
                elif n != 0:
                    s.append((h[n]/2)*(1-
cos((pi*m*omega)/beta[n])) + s[counterold])


v.append(((pi*h[n])/(beta[n]*(pi/180)*2))*sin((pi*m*omega)/beta[
n]))

a.append(((pi**2*h[n])/(beta[n]**2*(1/omega)**2*2))*cos((pi*m*om
ega)/beta[n]))
                j.append(((-
pi**3*h[n])/(beta[n]**3*(1/omega)**3*2))*sin((pi*m*omega)/beta[n
]))

                phi.append(atan((v[counter] -
epsilon)/(s[counter] + sqrt(Rprime**2 + epsilon**2))))

                v[counter] = v[counter]*omega*(pi/180)

                Fc.append(mass*( a[counter] +
2*zeta*omegan*v[counter] + omegan**2*(s[counter]-base) +
omegan**2*littledelta))

                Tc.append(
(Fc[counter]*cos(phi[counter])*v[counter])/(omega*(pi/180)) )

                P.append( abs(Tc[counter]*omega*(pi/180)) )
```

```
                Pa.append(
abs(Fc[counter]*cos(phi[counter])*v[counter]) )

                counter = counter + 1

        elif func[n] == -1:

            for m in arange(0,t[n],deltat):

                theta.append(counter*deltatheta)

                if n == 0:
                    s.append( h[n]*( (m*omega)/beta[n] -
(1/(2*pi))*sin((2*pi*m*omega)/beta[n])) + base)
                elif n != 0:
                    s.append( h[n]*( (m*omega)/beta[n] -
(1/(2*pi))*sin((2*pi*m*omega)/beta[n])) + s[counterold])

                v.append( (h[n]/(beta[n]*(pi/180))) * (1 -
cos((2*pi*m*omega)/beta[n])) )
                a.append(
((2*pi*h[n])/(beta[n]**2*(1/omega)**2)) *
sin((2*pi*m*omega)/beta[n]) )
                j.append(
((4*pi**2*h[n])/(beta[n]**3*(1/omega)**3)) *
cos((2*pi*m*omega)/beta[n]) )

                phi.append(atan((v[counter] -
epsilon)/(s[counter] + sqrt(Rprime**2 + epsilon**2))))

                v[counter] = v[counter]*omega*(pi/180)

                Fc.append(mass*( a[counter] +
2*zeta*omegan*v[counter] + omegan**2*(s[counter]-base) +
omegan**2*littledelta))

                Tc.append(
(Fc[counter]*cos(phi[counter])*v[counter])/(omega*(pi/180)) )

                P.append( abs(Tc[counter]*omega*(pi/180)) )
```

```
                Pa.append(
abs(Fc[counter]*cos(phi[counter])*v[counter]) )

                counter = counter + 1

        elif func[n] == 0:

            for m in arange(0,t[n],deltat):

                theta.append(counter*deltatheta)

                if n == 0:
                    s.append(0 + base)
                elif n != 0:
                    s.append(0 + s[counterold])

                v.append(0)
                a.append(0)
                j.append(0)

                phi.append(atan((v[counter] -
epsilon)/(s[counter] + sqrt(Rprime**2 + epsilon**2))))

                Fc.append(mass*( a[counter] +
2*zeta*omegan*v[counter] + omegan**2*(s[counter]-base) +
omegan**2*littledelta))

                Tc.append(
(Fc[counter]*cos(phi[counter])*v[counter])/(omega*(pi/180)) )

                P.append( abs(Tc[counter]*omega*(pi/180)) )

                Pa.append(
abs(Fc[counter]*cos(phi[counter])*v[counter]) )

                counter = counter + 1

        else:

            print "Your function number '{}' is invalid. Please
restart this code and try again.".format(func[n])
```

```
        counterold = counter-1

    cam = curve(radius = 0.005, display = scene)
    circle = curve(radius = 0.005, display = scene)
    origin = curve(radius = 0.01, display = scene)
    scalefactor = 1/base

    for n in arange(0,len(theta),1):

        cam.append(pos =
(scalefactor*s[n]*cos(theta[n]*(pi/180)),scalefactor*s[n]*sin(th
eta[n]*(pi/180)),0))
        circle.append(pos =
(cos(theta[n]*(pi/180)),sin(theta[n]*(pi/180)),0))
        origin.append(pos =
((1/100)*cos(theta[n]*(pi/180)),(1/100)*sin(theta[n]*(pi/180)),0
))

        sgraph.plot(pos=(time[n],s[n]))
        vgraph.plot(pos=(time[n],v[n]))
        agraph.plot(pos=(time[n],a[n]))
        jgraph.plot(pos=(time[n],j[n]))
        pgraph.plot(pos=(time[n],phi[n]*(180/pi)))
        Fcgraph.plot(pos=(time[n],Fc[n]))
        Tcgraph.plot(pos=(time[n],Tc[n]))
        Pgraph.plot(pos=(time[n],P[n]))
        Pagraph.plot(pos=(time[n],Pa[n]))

    pressuremax = max(phi)
    pressuremin = min(phi)
    pressure = max(abs(pressuremax),abs(pressuremin))

    print "Omega = {} deg/sec".format(rotate*omega)
    print "The maximum value of s is {} m".format(max(s))
    print "The maximum value of v is {} m/sec".format(max(v))
    print "The maximum value of a is {}
m/sec/sec".format(max(a))
    print "The maximum value of j is {}
m/sec/sec/sec".format(max(j))
```

```
    print "The maximum pressure angle is {}
deg".format(pressure*(180/pi))
    print ""

    for n in arange(0,zones,1):

        print "zone {}:".format(n)

        if func[n] == 1:
            print "s = ({})*(1 -
cos({}*theta)".format(h[n]/2,pi/beta[n])
            print "v =
({})*sin({}*theta)".format((pi*h[n])/(beta[n]*(1/omega)*2),pi/be
ta[n])
            print "a =
({})*cos({}*theta)".format((pi**2*h[n])/(beta[n]**2*(1/omega)**2
*2),pi/beta[n])
            print "j = ({})*sin({}*theta)".format((-
pi**3*h[n])/(beta[n]**3*(1/omega)**3*2),pi/beta[n])

        if func[n] == -1:
            print "s = ({})*(theta/{} -
({})*sin({}*theta)".format(h[n],beta[n],1/(2*pi),(2*pi)/beta[n])
            print "v = ({})*(1 -
cos({}*theta)".format(h[n]/(beta[n]*(1/omega)),(2*pi)/beta[n])
            print "a =
({})*sin({}*theta)".format((2*pi*h[n])/(beta[n]**2*(1/omega)**2)
,(2*pi)/beta[n])
            print "j =
({})*cos({}*theta)".format((4*pi**2*h[n])/(beta[n]**3*(1/omega)*
*3),(2*pi)/beta[n])

        if func[n] == 0:
            print "s = 0"
            print "v = 0"
            print "a = 0"
            print "j = 0"

        print ""

    print ""
```

```
    print "time (sec) \tdisplacement\tvelocity
\tacceleration\tjerk        \tPressure Angle\tForce on
Cam\tCamshaft Torque\tPower lost (Tc)\tPower Lost (Fc)"
    print "-----------------------------------------------------
-------------------------------------------------------------
-------------------------------------"
    print ""

    for n in arange(0,len(theta),1):
        print
"{:.10f}\t{:.10f}\t{:.10f}\t{:.10f}\t{:.10f}\t{:.10f}\t{:.10f}\t
{:.10f}\t{:.10f}\t{:.10f}".format(time[n],s[n],v[n],a[n],j[n],ph
i[n],Fc[n],Tc[n],P[n],Pa[n])
```