

To the Graduate Council:

I am submitting the enclosed thesis written by Jonathan W. Huber entitled “The Development and Verification of Three Matlab Analysis Applications Programmed Specifically for Engage Team Projects.” I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Engineering Science.

Christopher Pionke

Christopher Pionke, Major Professor

We have read this thesis and
recommend its acceptance:

J. Roger Parsons

Jaime Elaine Seat

Acceptance for the Council:

Anne Mayhew

Vice Provost and Dean of
Graduate Studies

(Original signatures are on file with official student records.)

The Development and Verification of Three Matlab Analysis Applications
Programmed Specifically for Engage Team Projects

A Thesis

Presented for the

Master of Science Degree

The University of Tennessee, Knoxville

Jonathan W. Huber

August 2003

Abstract

This thesis outlines the development of three analysis applications for use in the freshmen Engineering Fundamentals Program (*engage*) at the University of Tennessee. *Engage* teaches freshmen engineering mechanics concepts through a set of integrated components including Lecture, Analysis and Skills, Problem Session, Physical Homework, and Team Projects.

Presently, the students have limited access to any software analysis tools to aid the design process. The students have little experience writing the types of complex programs needed to analyze a particular project. Even more, they do not have the time. The applications provide a means for the students to perform an accurate analysis on their designs within the timeframe of the project deadlines.

The contents of this thesis outline the statics and dynamics team projects, a comparison of existing applications that model similar problems, an explanation of why Matlab was chosen as the programming language, a description of each application's features, verification problems, and application walkthrough tutorials.

Contents

Chapter 1 - Introduction	1
1.1 Background.....	1
1.1.1 <i>Engage</i> 's needs and requirements for pre-assembled applications.....	3
1.2 Team Projects.....	5
1.2.1 Statics Projects	5
1.2.2 Dynamics Projects	7
1.3 Review of Similar Applications.....	10
1.3.1 Truss solvers	10
1.3.2 Projectile motion.....	13
1.3.3 Summary of existing analysis applications.....	15
1.4 Application Objectives.....	15
1.4.1 Objectives for all applications	16
1.4.2 Objectives for the statics application	16
1.4.3 Objectives for the two dynamics applications	17
1.5 Thesis Objectives.....	18
 Chapter 2 - Development.....	 19
2.1 Application Details	19
2.1.1 Truss Analysis.....	19
2.1.1.1 Creating 2D and 3D capabilities in the same application	24
2.1.1.2 Error checking on for 2D and 3D models.....	25

2.1.2 Projectile Motion	26
2.1.2.1 Ramp and Spring Energy Dynamics Launch Velocity Calculation.....	27
2.1.2.2 Swing Energy Dynamics Launch Velocity Calculation	29
2.1.2.3 Projectile Motion Calculations from Launch Velocity	30
2.2.2.4 Modifying Plots	31
2.2 Deciding to use Matlab	31
2.3 Help Files	33
 Chapter 3 - Application Features	35
3.1 Truss Analysis Application.....	35
3.1.1 Toolbar Menu.....	37
3.1.2 Feature Menu	41
3.1.3 Modify Menu	42
3.1.4 Enter Values Menu	43
3.1.5 Analysis Menu	44
3.1.6 Axis Menu.....	45
3.1.7 Draw Area.....	47
3.1.8 View Display Menu	47
3.1.9 Help File.....	48
3.1.10 Internal Error Checking	49
3.2 Ramp Launched Projectile Motion Application	51
3.2.1 Toolbar	53

3.2.2 Time Increment Menu.....	55
3.2.3 Ramp Energy Menu	55
3.2.3.1 Angle calculations.....	57
3.2.3.2 Initial velocity up a ramp	58
3.2.3.3 Initial velocity down a ramp	60
3.2.3.4 Simple Initial Velocity Launch	62
3.2.4 Launch Energy Menu.....	62
3.2.5 Aim for Target Menu	65
3.2.6 Specify Launch Angle Menu	65
3.2.7 Drag Menu	67
3.2.8 Output to Figure.....	67
3.2.9 Analysis Button.....	69
3.2.10 Help File.....	71
3.3 Swinging Projectile Application	72
3.3.1 Toolbar	73
3.3.2 Properties Menu	77
3.3.3 Swing Energy Menu	77
3.3.4 Aim for Target Menu	79
3.3.5 Drag Menu	79
3.3.6 Output to Figure.....	81
3.3.7 Analysis Button.....	82
3.3.8 Help File.....	84

Chapter 4 - Verification and Tutorials	85
4.1 Truss Solver Application Verification	86
4.1.1 Hibbeler Example 6-1	86
4.1.1.1 Specify and Number Model Information	87
4.1.1.2 Enter Joint Information	88
4.1.1.3 Enter Member Data	90
4.1.1.4 Enter Constraint Data	91
4.1.1.5 Enter Force Data	93
4.1.1.6 Analyze	94
4.1.1.7 View Results	94
4.1.2 Hibbeler Example 6-2	96
4.1.3 Hibbeler Homework 6-30 / 6-31	100
4.1.4 Hibbeler Homework 6-62 / 6-63	105
4.2 Ramp Dynamics Verification Problems	110
4.2.1 Hibbeler Homework 14-28	111
4.2.2 Hibbeler Example 14-4	115
4.2.3 Hibbeler Homework 12-86	119
4.2.4 Boresi / Schmidt Example 14-7, Drag Verification	126
4.3 Swing Energy Application Verification	130
4.3.1 Homework 14-31	131
 Chapter 5 - Conclusions and Recommendations	 134

5.1 Conclusions	134
5.2 Recommendations	135
5.2.1 Truss Solver Application Modifications	135
5.2.2 Projectile Motion Application Modifications	135
5.2.3 Implementation Suggestions	136
References	137
Appendices	140
A EF 102 Spring 2003 Calendar and Team Projects	141
B Application Code	161
B.1 2D / 3D truss solver code	161
B.2 Ramp and Spring Energy Dynamics code	248
B.3 Swing Energy Dynamics code	281
Vita	314

List of Tables

Table 1.1. <i>Engage</i> Class Descriptions	2
Table 2.1 Simple and Complex Drag Calculations.....	31
Table 4.1 Hibbeler Example 6-1 Joint Information	87
Table 4.2 Hibbeler Example 6-1 Member Information	87
Table 4.3 Hibbeler Example 6-1 Constraint Information	88
Table 4.4 Hibbeler Example 6-1 Force Information.....	88
Table 4.5 Hibbeler Example 6-1 Reaction Forces	96
Table 4.6 Hibbeler Example 6-2 Joint Information	97
Table 4.7 Hibbeler Example 6-2 Member Information	97
Table 4.8 Hibbeler Example 6-2 Constraint Information	98
Table 4.9 Hibbeler Example 6-2 Force Information.....	98
Table 4.10 Hibbeler Example 6-2 Analysis Results	99
Table 4.11 Hibbeler Homework 6-30 / 6-31 Joint Information.....	101
Table 4.12 Hibbeler Homework 6-30 / 6-31 Member Information	102
Table 4.13 Hibbeler Homework 6-30 / 6-31 Constraint Information	102
Table 4.14 Hibbeler Homework 6-30 / 6-31 Force Information	102
Table 4.15 Hibbeler Homework 6-30 / 6-31 Results	104
Table 4.16 Hibbeler Homework 6-62 / 6-63 Joint Information.....	106
Table 4.17 Hibbeler Homework 6-62 / 6-63 Member Information	107

Table 4.18 Hibbeler Homework 6-62 / 6-63 Constraint Information.....	107
Table 4.19 Hibbeler Homework 6-62 / 6-63 Force Information	107
Table 4.20 Hibbeler Homework 6-62 / 6-63 Results.....	110
Table 4.21 Ramp Energy Application Data.....	112
Table 4.22 Hibbeler homework 14-28 Time Increment Output	114
Table 4.23 Ramp Energy Application Output	115
Table 4.24 Hibbeler Example 14-4 data entered into application	116
Table 4.25 Hibbeler Example 14-4 application results.....	118
Table 4.26 Hibbeler Homework 12-86 data entered into application.....	120
Table 4.27 Hibbeler Homework 12-86 Time Increment and Angle Precision Results ..	125
Table 4.28 Boresi / Schmidt Example 14-7 data entered into application.....	127
Table 4.29 Boresi / Schmidt Example 14-7 Results	129
Table 4.30 Boreshi / Schmidt Example 14-7 Application, Excel, and Text Answer	130
Table 4.31 Homework 14-31 data for application.....	132
Table 4.32 Hibbeler Homework 14-31 application results.....	133

List of Figures

Figure 1.1 Typical Truss Project.....	5
Figure 1.2 Top View of Testing Area.....	8
Figure 1.3. Dynamics Team Project Timing Mechanism (left) and Launcher (right).....	8
Figure 1.4. Dynamic Team Project Top View.....	8
Figure 1.5. University of Minnesota 2D Truss Solver.....	11
Figure 1.6 West Point Bridge Designer.....	12
Figure 1.7. Web-based Projectile Motion Application.....	14
Figure 1.8. Excel Based Projectile Motion User Input.....	14
Figure 1.9. Excel Based Projectile Motion Data Plots.....	14
Figure 2.1 Basic truss and forces at each joint.....	21
Figure 2.2 Arranged unknowns of the basic truss.....	21
Figure 2.3 Basic truss arranged into $[A]\{x\} + \{b\} = \{0\}$	23
Figure 2.4 Basic truss results.....	23
Figure 2.5 Ramp and Spring Energy Typical Analyses.....	28
Figure 2.6 Swing Energy Description.....	29
Figure 3.1 <i>Engage</i> Truss Solver.....	36
Figure 3.2 Truss Solver Toolbar.....	37
Figure 3.3 Truss Solver, File Menu.....	37
Figure 3.4 Truss Solver, View Menu.....	38

Figure 3.5 Truss Solver, Display Menu	38
Figure 3.6 Truss Solver, Error Checking Menu.....	40
Figure 3.7 Truss Solver, Help Menu.....	40
Figure 3.8 Truss Solver, Feature Menu	41
Figure 3.9 Feature, Modify, and Enter Values Menus.....	41
Figure 3.10 Truss Solver, Modify Menu	43
Figure 3.11 Truss Solver, Enter Values Menu.....	43
Figure 3.12 Truss Solver Analysis Menu	44
Figure 3.13 Truss Solver, Model Details Menu.....	45
Figure 3.14 Truss Solver, Axis Menu.....	46
Figure 3.15 Full Truss (left) and Zoomed View (right).....	46
Figure 3.16 Truss Solver, Draw Area	47
Figure 3.17 Truss Solver, View Menu Output.....	47
Figure 3.18 Truss Solver, Help File.....	48
Figure 3.19 Truss Solver, Non-numeric Value Error	50
Figure 3.20 Truss Solver, Duplicate Entry Error.....	50
Figure 3.21 Truss Solver, Non-existing Entry Error.....	51
Figure 3.22 Ramp and Spring Launching Application	52
Figure 3.23 Ramp and Spring Launching Application, Toolbar	53
Figure 3.24 Ramp and Spring Launching Application, Display Menu	54
Figure 3.25 Ramp and Spring Launching Application, Sample Output.....	54
Figure 3.26 Ramp and Spring Launching Application, Unit Selection Menu.....	55

Figure 3.27 Ramp and Spring Launching Application, Unit Display.....	56
Figure 3.28 Ramp and Spring Launching Application, 0.5s Time Increment.....	56
Figure 3.29 Ramp and Spring Launching Application, 0.1s Time Increment.....	57
Figure 3.30 An upward ramp and positive dy value	58
Figure 3.31 A downward ramp and negative dy value	58
Figure 3.32 Ramp and Spring Launching Application, Ramp Upward.....	59
Figure 3.33 Ramp and Spring Launching Application, Ramp Upward Results.....	60
Figure 3.34 Ramp and Spring Launching Application, Ramp Downward.....	61
Figure 3.35 Ramp and Spring Launching Application, Specified angle (left) and Ramp Downward Results (right).....	61
Figure 3.36 Ramp and Spring Launching Application, Simple Velocity Analysis	62
Figure 3.37 Ramp and Spring Launching Application, Simple Velocity Results	63
Figure 3.38 Ramp and Spring Launching Application, Launch Energy Menu	63
Figure 3.39 Ramp and Spring Launching Application, Combination Launch Output	64
Figure 3.40 Ramp and Spring Launching Application, Spring Launch Output	64
Figure 3.41 Ramp and Spring Launching Application, Target Menu	65
Figure 3.42 Ramp and Spring Launching Application, With (left) and Without (right) Target Menu.....	66
Figure 3.43 Ramp and Spring Launching Application, Specific Launch Angle Menu...	66
Figure 3.44 Ramp and Spring Launching Application, With (left) and Without (right) Specific Launch Angle.....	66
Figure 3.45 Ramp and Spring Launching Application, Drag Menu.....	67

Figure 3.46 Ramp and Spring Launching Application, Figure Output.....	68
Figure 3.47 Ramp and Spring Launching Application, Data Output	68
Figure 3.48 Ramp and Spring Launching Application, Analyze Button.....	69
Figure 3.49 Ramp and Spring Launching Application, Analyze Output.....	70
Figure 3.50 Ramp and Spring Launching Application, Help File	71
Figure 3.51 Swing Launching Application.....	73
Figure 3.52 Swing Launching Application, Toolbar	74
Figure 3.53 Swing Launching Application, Display Menu	74
Figure 3.54 Swing Launching Application, Output.....	75
Figure 3.55 Swing Launching Application, Units Selection Menu	75
Figure 3.56 Swing Launching Application, Units Display.....	76
Figure 3.57 Swing Launching Application, Properties Menu	77
Figure 3.58 Swing Launching Application, Swing Energy Menu.....	78
Figure 3.59 Swing Launching Application, User Input.....	78
Figure 3.60 Swing Launching Application, Aim for Target Menu	79
Figure 3.61 Swing Launching Application, With (left) and Without (right) Target Menu	80
Figure 3.62 Swing Launching Application, Drag Menu.....	80
Figure 3.63 Swing Launching Application, Figure Output	81
Figure 3.64 Swing Launching Application, Data Output	82
Figure 3.65 Swing Launching Application, Analyze Button.....	83
Figure 3.66 Swing Launching Application, Analysis Output.....	83

Figure 3.67 Swing Launching Application, Help File	84
Figure 4.1 Hibbeler Example 6-1.....	86
Figure 4.2 Adding joint 1 to the model.....	89
Figure 4.3 Results of adding joint 2 (left) and joint 3 (right) to the model	89
Figure 4.4 Steps to adding member 1 to the model	90
Figure 4.5 Results of adding member 2 (left) and member 3 (right) to the model	91
Figure 4.6 Steps to adding a new constraint	92
Figure 4.7 Results of adding all model constraints.....	92
Figure 4.8 Steps to adding a new force.....	93
Figure 4.9 Example 6-1 complete model.....	95
Figure 4.10 Example 6-1 analysis.....	95
Figure 4.11 Hibbeler Example 6-2.....	97
Figure 4.12 Hibbeler Example 6-2 entered into truss solver	98
Figure 4.13 Hibbeler Example 6-2 Analysis Results.....	99
Figure 4.14 Figure for Hibbeler homework problems 6-30 and 6-31.....	100
Figure 4.15 Hibbeler Homework 6-30 / 6-31 entered into truss solver	103
Figure 4.16 Hibbeler Homework 6-30 / 6-31 analysis results.....	103
Figure 4.17 Figure for Hibbeler homework problems 6-62 and 6-63.....	106
Figure 4.18 Hibbeler Homework 6-62 / 6-63 entered into truss solver	108
Figure 4.19 Hibbeler Homework 6-62 / 6-63 analysis results	109
Figure 4.20 Figure for Hibbeler homework 14-28.....	112

Figure 4.21 Ramp angle CCW from the x-axis	113
Figure 4.22 Hibbeler homework 14-28 data into Ramp Energy Menu	114
Figure 4.23 Application results for Hibbeler homework 14-28.....	114
Figure 4.24 Hibbeler Example 14-4, Platform, unloaded (left) and loaded (right)	116
Figure 4.25 Hibbeler Example 14-4 Time Increment, Ramp Energy, and Spring Energy Menu Settings	117
Figure 4.26 Hibbeler Example 14-4 application results	118
Figure 4.27 Hibbeler Homework 12-86 Figure	120
Figure 4.28 Possible launch angles CCW from x-axis	122
Figure 4.29 Hibbeler Homework 12-86 “Low Arc” solution.....	122
Figure 4.30 Hibbeler Homework 12-86 “High Arc” solution	123
Figure 4.31 Hibbeler Homework 12-86 “High Arc” solution focused on target.....	123
Figure 4.32 Hibbeler Homework 12-86 “High Arc” solution, with increased launch angle precision.....	124
Figure 4.33 Hibbeler Homework 12-86 Direct Path missing target	126
Figure 4.34 Boresi / Schmidt Example 14-7 Figure	127
Figure 4.35 Boresi / Schmidt Drag Example 14-7 User Entry	128
Figure 4.36 Boresi / Schmidt Example 14-7 Output.....	129
Figure 4.37 Hibbeler Homework 14-31 Figure	132
Figure 4.38 Data for Hibbeler Homework 14-31.....	133
Figure 4.39 Hibbeler Homework 14-31 application results.....	133

Chapter 1 - Introduction

The Engineering Fundamentals Program at the University of Tennessee, known as the *engage* program, requires a package of analysis applications.. Three applications make up the package; a 2D and 3D truss analysis package and two specialized projectile motion packages. Each application corresponds to an *engage* team project, utilizes Matlab as a programming language, uses a graphical user interface, and includes complete help files and tutorials.

1.1 Background

The *engage* freshmen-engineering program at the University of Tennessee consists of two six-hour classes, EF 101 and EF 102, taught over the Fall and Spring semesters, respectively. The *engage* curriculum covers material in basic physics, statics, dynamics, computer programming, laboratory experiments, and team projects in an integrated manner [1].

Each semester is divided into a series of Modules. A Module is a packet of individual components with common topical themes including a general Lecture, Analysis and Skills, Problem Session, Physical Homework, and Team Project Time. The Modules are usually two to three weeks in length. A component breakdown of *engage* is shown in Table 1.1. Each component within EF 101 and EF 102 contributes to a different portion of the students' education. The course outline for Fall and Spring 2003, including the descriptions of all the student project assignments, is shown in Appendix A.

Table 1.1. *Engage* Class Descriptions

<i>Engage</i> Component	Description
Lecture	Introduces the mechanics concepts, formulas, and general background for a particular topic.
Analysis and Skills / Problem Session	Drafting, CAD, and computer programming during the Fall. Statics and dynamics with some computer programming in the Spring.
Physical Homework	Student laboratory of the mechanics concepts taught in Lecture.
Team Project Time	Students divide into teams and then design and build projects to perform a specified task. Usually, two or three team projects a semester.

The Lectures take place in an auditorium and occurs three times a week. In the Lectures, the students are taught the mechanics concepts for the Module. The Analysis and Skills sessions teach concepts in drafting, computer drafting, and computer programming. The Problem Sessions are mainly for the students to solve practice problems selected from the textbooks, which correspond to the current Module and are directly related to the Lecture. Physical Homework Sessions are hands-on laboratory experiments that reinforce the concepts taught in Lecture. The Team Project Time divides a thirty-student class into teams of five students to develop teaming, communication, and design skills. Generally, team projects span multiple Modules and reinforce the concepts taught in Lecture.

Engage organizes and integrates each of the students' activities. The integration can be as simple as introducing the concept of a coordinate system in General Lecture and then reinforcing that concept as a drafting tool in Analysis and Skills, or as complex as

learning the mathematical technique to solve 2D truss problems, then writing a Matlab [2] program to solve more complicated 2D truss problems, performing related experiments in Physical Homework attempting to determine and measure the member forces in a small truss, and all the while building a complex 3D bridge for a team project.

1.1.1 *Engage*'s needs and requirements for pre-assembled applications

In Analysis and Skills, the students learn programming skills using Matlab as a language. The Matlab programs are designed to answer simple homework problems. Near the end of the EF 101 and during EF 102, the assigned Matlab problems are more advanced and are often related to the team projects. With some modifications to their code from the assignment, the students can perform the analysis for their team project. However, due to time restrictions, the students don't always have an opportunity to make the necessary modifications to their code, analyze their design, implement any changes, and finish their projects by the deadline. When the code is ready to analyze their model, the current project is completed and they start a new project.

This thesis outlines the development of three Matlab applications to aid students' analyses of the *engage* team projects. To help the students use each application, comprehensive help files and tutorials are included. The applications include error checking to monitor the validity of the user inputs. Also, they are each in a window based Graphical User Interface (GUI) to make them more user-friendly.

At present, in EF 102, students are required to design projects to perform specified tasks, such as building a bridge (statics) or launching a projectile towards a

target (dynamics). The projects are helpful in the students' understanding of the mechanics concepts associated with the design issues being taught in the Module, but the students have limited options when attempting to analyze their design before the completion of the project.

With a prepared package of applications, the students will have time to analyze their design and make any necessary changes to their project. The applications reinforce the notion of tool usage, in particular, Matlab, to complete a specific step in the problem solving process.

Additionally, the applications may improve the morale surrounding Matlab assignments. As part of teaching Matlab, the students are taught to write organized, generic, and commented code for the "mysterious random user". When they use the applications for the first time, the students take on the role of that "random user". From this perspective, the students learn why it is important to write organized, generic, and commented code. Also, they will benefit from the help files, tutorials, and error checking supplied for the bungling "mysterious random user".

More importantly, the students interact with the versatility of Matlab. While the students are learning the programming language, they don't comprehend a complete picture of its capabilities. The classroom examples are limited because of time. Also, the examples are meant to discuss a particular set of features. With these analysis applications, the students use Matlab to quickly and easily solve complex problems. Furthermore, the students recognize the usefulness of the computer solutions and the effectiveness of Matlab as a programming language to solve an analytical problem.

1.2 Team Projects

Engage has a variety of team projects for the students. The design projects are divided into two basic categories, statics and dynamics. Appendix A provides examples of project descriptions from EF 102 in the 2003 Spring semester.

1.2.1 Statics Projects

For the statics projects in EF 102, the students usually need to build a bridge, which (hopefully) withstands a specified load. Typically, the bridge must span a distance of roughly eight feet with the builders standing on the bridge to supply the load. A typical student built truss is shown in Figure 1.1.



Figure 1.1 Typical Truss Project

As a project restriction, the students are only allowed to perform testing on individual truss members and never the entire truss. This requirement places a premium on the ability to calculate the member forces in their entire design. The combination of member testing and 3D analysis in the provided thesis statics application will provide feedback as to where the load is greater than the member is expected to carry for the design.

The students are taught the method of joints to solve for member and reaction forces through a series of lectures in Modules 1 and 2 during EF 102. After the lectures, the students have been introduced to the theory, but they do not understand the material to the degree to convert their knowledge into working code to solve the member and reaction forces in their design. Also, they don't have the time to write a complete 3D truss solver.

At present, the students have access to a simple 2D truss solving Matlab code and instructions of how the code operates, but it requires the model information to be hard coded into variables. This method is an effective way to apply method of joints for simple trusses, but it isn't efficient for larger models. The code does not use a GUI and has a sharp learning curve because of the complex data entry methods.

The visual interface provides an interactive means of data entry. The students still need to know the coordinates of the joints, member connectivity, applied forces, and constraints for the design. With the proposed application, the model is built as a visual object, instead of a listing of hard coded variables. The truss solver application will provide a GUI for the numerical analysis.

The visual, user-friendly environment will lower the learning curve associated with the standard hard coded function. With an intuitive, visual application, the students who usually avoid Matlab are capable of performing an analysis. Optimistically, the students apply the analysis into their design and begin to understand the method of joints mechanics concepts.

1.2.2 Dynamics Projects

As mentioned in a previous section, each of the projects corresponds to a particular set of dynamics Modules. The following is a sample team project called *Critter Conker* that spans EF 102 Modules 3-5. A detailed project description is listed in Appendix A.

The students are required to design a projectile (paintball) launcher to hit a moving target. As shown in Figure 1.2, the projectile must be launched over a barrier precisely at the correct moment to hit the *critter* moving along a transverse path. The students are informed on test day which particular path the *critter* will follow. They have a specified time limit to make the necessary changes to their launcher.

One team's launcher is shown in Figures 1.3 and 1.4. This design consists of a marble moving down a series of ramps, a trigger mechanism, and a rubber band launcher. The ramp acts as a timing mechanism to delay the marble hitting the triggering device. With a bit of luck, the projectile lands at the intended location and time as the moving *critter* crosses the centerline.

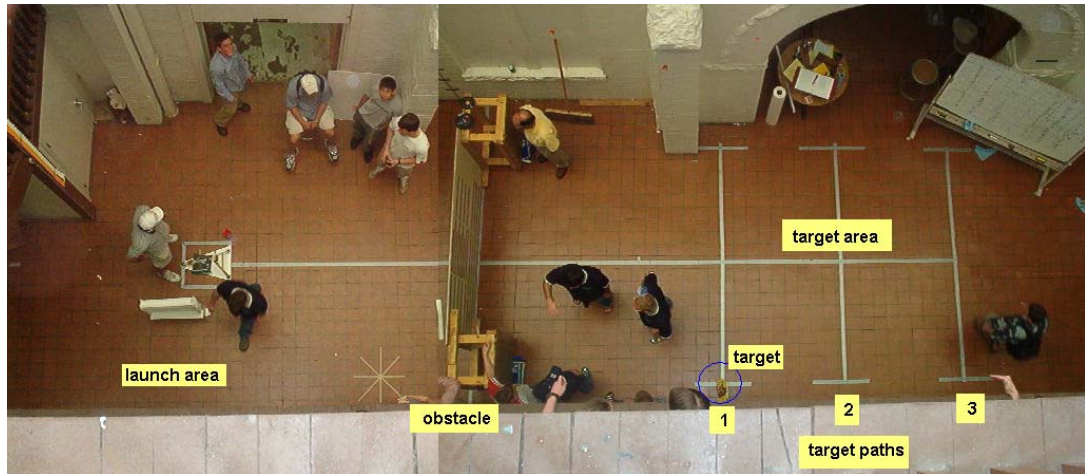


Figure 1.2 Top View of Testing Area



Figure 1.3. Dynamics Team Project Timing Mechanism (left) and Launcher (right)

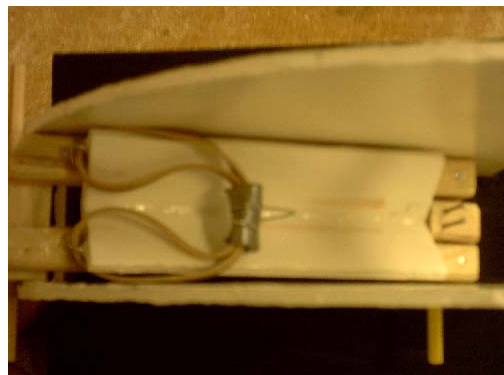


Figure 1.4. Dynamic Team Project Top View

This project requires the students to design for two steps. First, they need to calculate time the projectile is in the air. This includes calculating the launch speed, height, angle, and time. Second, they need to calculate the time delay needed before the launch, using the speed and path of the *critter*. Proper implementation of the calculations into the design will result in the projectile hitting the moving *critter*. *engage* This project is typical of other projectile motion *engage* dynamics related team projects.

The *Critter Conker* is one example of the typical dynamics projects assigned. In general, the dynamics projects require the students to launch a projectile towards a target using a variety of launching mechanisms. The launching mechanism varies from a vehicle sliding down a ramp then launching a spring loaded projectile, a vehicle moving up a ramp with a particular velocity from a loaded spring, or a swinging vehicle that launches a projectile at a particular angle.

These applications will account for a simple velocity vector, swing, ramp, and spring launching energies. This range of launching methods makes the applications compatible with the typical dynamics oriented team projects. The two applications account for most launching methods that have been used in *engage*'s past.

Much like the statics design tasks, the students are introduced to the theory but are not capable of programming complex applications, especially not in the allowed time period. The applications allow the student to focus on the analysis, not creating the application to obtain the analysis. The students have access to code from their related assignments, but this code requires hard coded data and is not user-friendly.

1.3 Review of Similar Applications

The existing statics and dynamics applications are programmed for specified analyses. For *engage* to use an existing application, it must match to the typical *engage* team projects. For existing truss solvers, two typical 2D truss solvers and two typical 3D stress analysis solvers are examined. For existing projectile motion programs, a simple web-based application and a general Excel based spreadsheet are considered for use with the *engage* team projects.

1.3.1 Truss solvers

The students need a means to quickly analyze their 2D and 3D truss-bridge designs. For 2D models, there are many options available, such as a simple method of joints solver or the West Point Bridge Designer [3]. For 3D models, there are finite element analysis packages, which provide stress analysis of the entered model, such as Cosmos [5] or Ansys [6]. To apply to the team projects, the provided application must be able to build a 2D or 3D model and calculate the member and reaction forces associated with the model.

A simple 2D truss solver, written by Professor T. W. Shield at the University of Minnesota, is shown in Figures 1.5 [4]. This 2D solver is similar to other available 2D truss solvers. While this program is an easy means of calculating member forces, the program neither has a user-friendly environment nor solves 3D models.

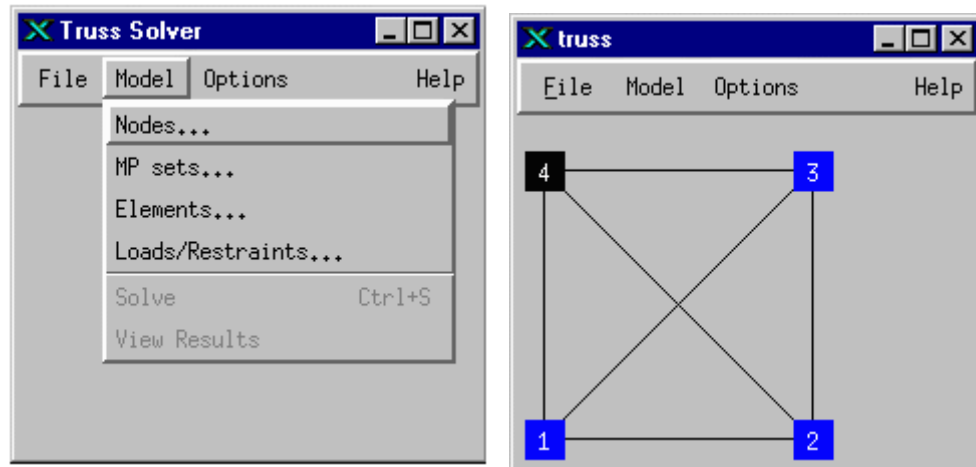


Figure 1.5. University of Minnesota 2D Truss Solver

To enter data in the mentioned 2D truss solver, the user selects Nodes, MP Sets (material property settings), Elements, or Loads / Restraints. Once the option has been selected, a menu appears for the user to enter the specific data. The 2D model is built in the same window as the program. The solver takes the material data and calculates a deflection on the member from the load conditions. However, it can only solve simple 2D models and previous entries cannot be modified. Also, the model's display does not intuitively depicting the features. For instance, a force is displayed by changing the numeric background color from blue to black (as with node 4), not as a directional arrow.

Another truss solver is the West Point Bridge Designer (WPBD), shown in Figure 1.6. The user builds a bridge using a selection of beams. If the load on the bridge, caused by a truck driving over the bridge, creates a member force greater than the beam can carry, then the program reports the failure.

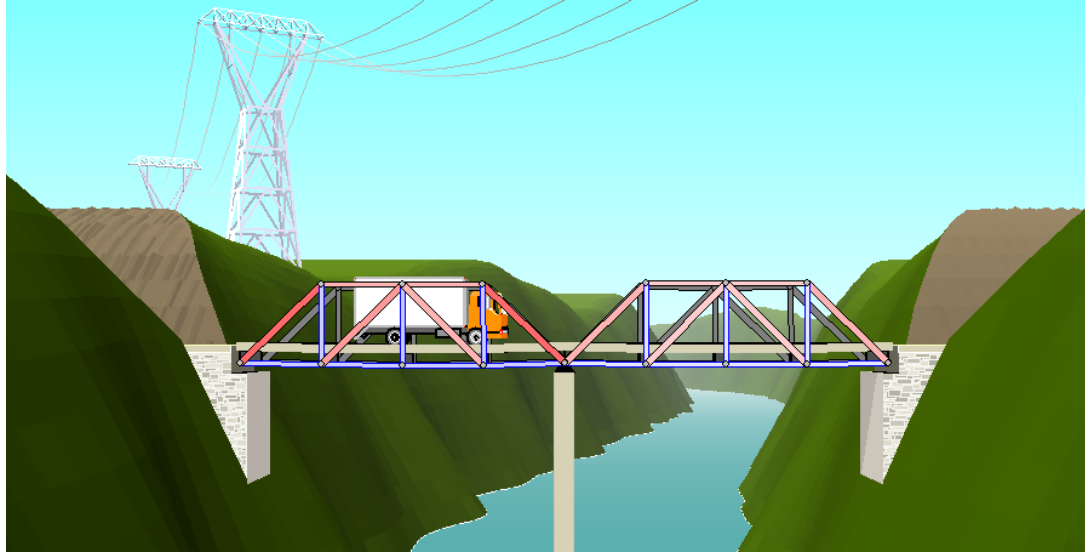


Figure 1.6 West Point Bridge Designer

The WPBD is capable of calculating the forces in each member of the truss, but it only solves 2D models. Also, by considering beams of standard materials, it doesn't account for the common materials used by *engage*. For instance, wood beams are not part of the WPBD.

The 3D solvers are capable of solving member and reaction forces in trusses. Analysis programs, such as Cosmos or Ansys, are capable of solving extremely complex 3D problems, but the students would need to be trained on the software and become familiar with the theory behind the analysis. For an application to be a good modeling tool for the students, they must have some knowledge of the mathematical technique. The students may be capable of building a model, but they will not understand the mechanics concepts behind the software. Second, the cost of licensing the software in a computer lab would be a tremendous financial expenditure.

The proposed static application will fulfill the analysis needs for the team truss project. Also, the students are taught method of joints and by utilizing that technique, the students understand of the mathematics used by the application.

1.3.2 Projectile motion

There are two applications outlined in this section. Each application is typical of most available projectile motion programs. One is a web-based application and the other is an Excel based application. Each of these applications is capable of solving simple projectile motion calculations.

One particular web-based application is from University of Oregon, Physics Department [7], in Figure 1.7. It solves simple projectile motion paths and accounts for drag. This application has a limited user input to specify launching types and the user can't reposition the target. This application does not have the flexibility to apply to a wide range of team projects, because it cannot account for any type of launching methods. Obviously, the purpose of this application is to provide a simple projectile motion analysis, and will not suit the more complex *engage* dynamics team projects.

The user can customize the Excel program from George Mason University, Physics and Astronomy Department [8] (see Figure 1.8). With an Excel file, the user can enter new columns for other calculations, like drag. One benefit to this program over the web-based application is the data output of the results. If a new plot needs generated, then the user can modify the Excel file to display the proper information, as in Figure 1.9.

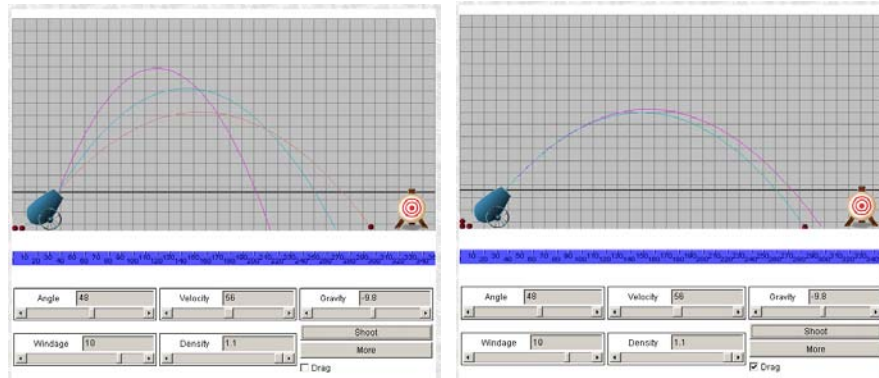


Figure 1.7. Web-based Projectile Motion Application

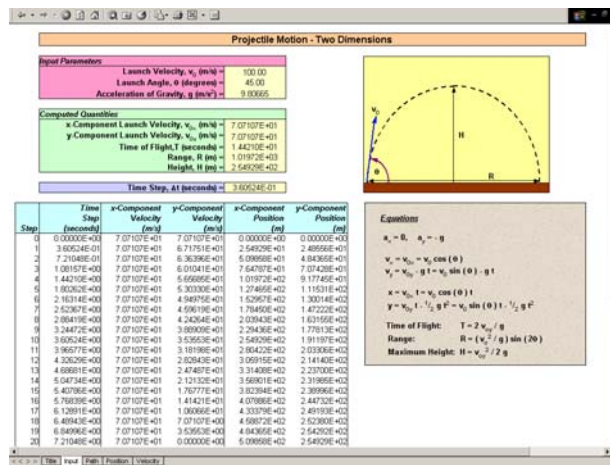


Figure 1.8. Excel Based Projectile Motion User Input

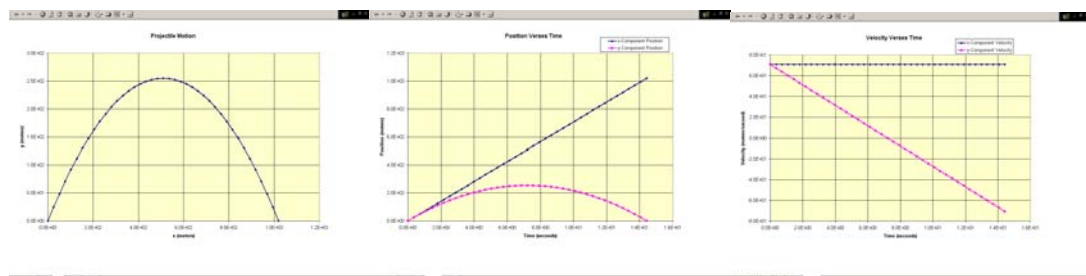


Figure 1.9. Excel Based Projectile Motion Data Plots

If the user needs to calculate the launch velocity from a launching mechanism, then modifications can be made to the file.

This application includes the user customization and data output necessary for the *engage* students' analytical needs. For the *Critter Conker* project, the students use the raw data to analyze the flight path and determine the launch time. However, *engage* doesn't teach Excel. Therefore, the students would be required to learn new software to analyze their projectile motion projects.

1.3.3 Summary of existing analysis applications

While the existing applications effectively perform one particular analysis, they do not efficiently integrate with *engage*. The analysis of the software is not specialized to *engage*'s team projects or curriculum. Additionally, most of the existing applications cannot be modified to meet the team project's analysis needs. Even more, none of the existing projectile motion applications are programmed in Matlab with a window based graphical user interface.

1.4 Application Objectives

The objectives for the Matlab applications are divided into three categories: general objectives for all applications, objectives for the statics application, and objectives for the two dynamics applications.

1.4.1 Objectives for all applications

- Must use a graphical user interface (GUI)
- Must have an intuitive operation
- Include error checking
 - Verify input
 - Verify model validity
- Help file
 - Each button and user data entry is described
 - Includes a example walkthrough tutorials

1.4.2 Objectives for the statics application

- The truss solver must solve for member forces using method of joints
 - Display Results
 - Members are red or blue for tension or compression
 - Thickness of member is proportional to the magnitude of the member forces
- The truss solver must allow 2D and some 3D models
- Implement a user-friendly environment
 - View Options
 - Zoom
 - Select Top, Front, Side, or Isometric view.

- Rotation of View
 - Modify previously entered values
 - Save / Open / Edit models

1.4.3 Objectives for the two dynamics applications

- The applications must account for four launching methods
 - Specific launch velocity and launch angle projectile motion
 - A vehicle sliding up or down a ramp and launched into the air at the calculated exit velocity from the energy exchange.
 - A release of spring energy and the end of the ramp
 - A swinging vehicle releasing a projectile
- Have the option to account for drag
 - A simple initial percentage reduction in energy
 - An iterative drag force calculation based on user entry coefficients

By satisfying these objectives, the applications reinforce concepts that are taught by *engage* to the students. The students are using numerical tools to solve an analytical problem. In particular, they are using Matlab to solve a mechanics problem. The visual interface provides a more intuitive environment for the students to solve problems. The analysis applications save the students' time, and reinforce the basic *engage* mechanics concepts.

1.5 Thesis Objectives

The focus of this thesis is to create applications for use in the *engage* program. Matlab is the preferred programming language because of its ability to solve analytical problems, and it is the programming language taught as a part of Analysis and Skills. Chapter 2 presents a comparison of Matlab to other programming languages and details Matlab's analytical advantages. Chapter 3 provides a description of each feature contained in the completed applications. Chapter 4 outlines the verification of results and application walkthrough tutorials to guide to user through the application's capabilities. Chapter 5 offers suggestions for implementation and improvements to the created applications. Appendix A is the *engage* 2003 Spring semester calendar and the team project descriptions. Appendix B contains the code for the created applications.

Chapter 2 - Development

2.1 Application Details

For all the applications, the user input is within the main user window GUI. This format keeps students from searching for a particular menu to enter a piece of data into their model. Instead, the user input is through one interface, which is in front of the students at all times. The toolbar is used for file management, view options, and a few user specific features such as, unit system selection, error checking, and help file access. All functions in the main user window are specific to each application and easy to access.

2.1.1 Truss Analysis

The truss analysis will apply the method of joints to solve for the member and reaction forces. Once calculated, the vector and magnitude of the force can be displayed in two different formats, either onto the screen as tensile or compression (red or blue) with the reaction forces as green vector arrows, or as text output containing a table of results. The text output is saved as a file and is also displayed in the Matlab Command Window.

To easily pass model variables between functions, the joint, member, constraint, and force variables are global. By allowing the key variables to be passed without any special commands, the global variables make a student customized or an additional provided analysis function easier to implement. Making a variable global is not a

programmer-preferred method. If a programmer is not careful, then the global variables could overlap with a different variable of the same name. Matlab has a built-in defense for this issue. The global variable only exists inside a function where it has been initialized. Hence, if students want access to the variable in the Matlab Command Window, they must first initialize it from the Command Window. The variable doesn't exist in a function until it has been initialized. Once the variable is available, it contains the values of the global variable and any changes made by the user affect the global variable.

After the joints, members, constraints, and forces have been entered into the application, the next step is to turn the global variables into a coefficient matrix for Matlab to solve. The students learn this process in EF 102, Module 2 Lecture 3 [9].

Building the matrix starts by drawing the free body diagrams of each joint, shown in Figure 2.1. Joint A is pinned to a wall, this pin constraint limits motion on the X and Y axes and creates reaction forces for both constraints, shown as A_y and A_x on the figure. Joint C is a roller constraint and creates the force C_x . This type of constraint limits motion perpendicular to the contact surface, in this case, the X-axis.

Because this truss is in static equilibrium, each joint is also in static equilibrium. Figure 2.2 shows the equilibrium equations for each joint. The “c” and “s” are abbreviations for cosine and sine, respectively. The variables are arranged to group the constraint forces (reaction forces) and member forces. The equilibrium equations for each joint and axis are stored in separate rows. The arrangement is meant to separate the unknowns of the joint equilibrium equations into the same columns. This arrangement

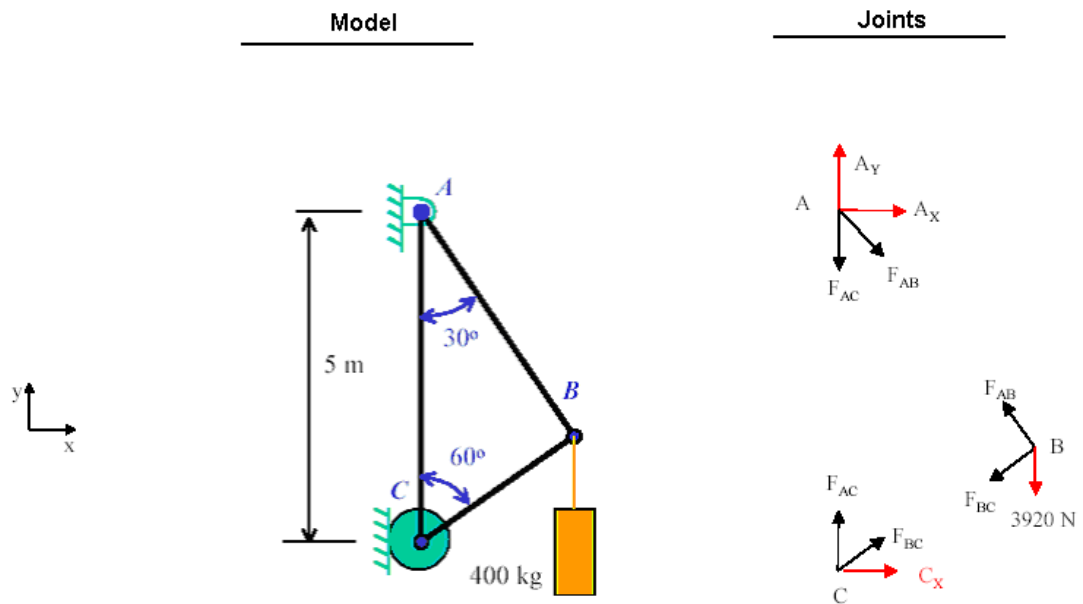


Figure 2.1 Basic truss and forces at each joint

joint A	$\Sigma X =$	$+ A_X$	$+ F_{AB} \cos(300)$	$= 0$	
	$\Sigma Y =$	$+ A_Y$	$+ F_{AB} \sin(300) + F_{AC} \sin(270)$	$= 0$	
joint B	$\Sigma X =$		$+ F_{AB} \cos(120)$	$+ F_{BC} \cos(210)$	$= 0$
	$\Sigma Y =$		$+ F_{AB} \sin(120)$	$+ F_{BC} \sin(210) + 3920 \sin(270)$	$= 0$
joint C	$\Sigma X =$	$+ C_X$		$+ F_{BC} \cos(30)$	$= 0$
	$\Sigma Y =$		$+ F_{AC} \sin(90)$	$+ F_{BC} \sin(30)$	$= 0$

Figure 2.2 Arranged unknowns of the basic truss

results in a matrix of coefficient data. The coefficients determine how much of the force is applied on each axis from the equilibrium equations for a particular joint. For instance, if a member force was 30° counter-clockwise from the x-axis, then the x coefficient would equal the $\cos 30^\circ$ and the y coefficient would equal the $\sin 30^\circ$.

After the equilibrium equations have been arranged, the member and reaction forces are separated from the coefficient data, as shown in Figure 2.3. The unknown member and reaction forces are stored in vector $\{x\}$. In the form $[A]\{x\} + \{b\} = \{0\}$, the equation is in equilibrium. Because the truss is in static equilibrium, the combination of all member, reaction, and external forces is equal to zero. Matrix $[A]$ corresponds to the coefficients that describe the free body diagrams for each joint. The first three columns relate to the constraints A_x , A_y , and C_x . Joint B has no constraints, so, no value exists in these columns for that joint. The remaining columns relate to the geometry coefficients for members AB, AC, and BC. The Vector $\{x\}$ is the list of unknown values. The Vector $\{b\}$ contains all external forces, in this case, the lone force in the Y direction at Joint B.

To solve for the unknown variables in vector $\{x\}$, the equation $[A]\{x\} + \{b\} = \{0\}$ needs to be rearranged to $[A]\{x\} = \{-b\}$. The vector $\{b\}$ is negative because it was moved to the other side of the equation. With the data entered into Matlab, a solution can be found, shown in Figure 2.4. Calculating the solution for the unknown member and reactions forces vector $\{x\}$ in Matlab is executed by a *simple single command* $x = A \setminus -b$.

2.1.1.1 Creating 2D and 3D capabilities in the same application

One alternative is to create separate applications for 2D and 3D analysis.

However, the mathematics for building the coefficient matrix is the same for both 2D and 3D models. Hence, adding the capability to solve for 2D and 3D is simple to implement.

When the application is determining the coefficient for a particular member, it checks the x, y, and z vector components relative to the scalar length to determine the angle of the

member force vector, $Coefficient(x, y, z) = \frac{V\hat{i} + V\hat{j} + V\hat{k}}{\|\vec{V}\|}$. When the student builds a 2D

model, one vector component will be equal to zero, because the member should not have a length component off of the specified 2D plane.

Another benefit of the dual dimensional capabilities is the ability to analyze a model in 2D, make a few modifications, and analyze the same model in 3D. A setting on the main view area controls the type of dimensional analysis. When analyzing a truss, the application builds the matrix for a 3D model. For a 2D analysis, after checking the dimension and plane, the unnecessary entries are removed from the matrix. For instance, if the model was built on the XZ plane, then all entries for the Y vectors are removed. The same is true for the XY and YZ planes.

The matrix entry removal process was extremely simple to implement. The removal starts with the last row and column and moves towards the first. This method does not affect the numbering system of the rows and columns. If the Y components of Joint 1 are removed first, then it affects the row and column number of Joint 2.

While a 2D truss can be built on any plane, the front view is set to the XZ plane. The coordinate system is the default XYZ coordinate system for Matlab, and is identical to the default coordinate system used in Mechanical Desktop [10], the CAD package taught in the Analysis and Skills portion of EF 101 and EF 102.

Once the model is built, clicking a button initiates the analysis function. There are two calculations related to drawing each member force. First, the magnitude of all the applied, member and reaction forces is compared to the maximum value of all the forces. Based on the global maximum, each member is drawn thicker for high forces, thinner for smaller forces. Second, the members are blue for a compressive (negative value) load, red for a tension (positive value) load. The value of each member force is drawn in the center of the member. If the students want to view a list of forces, then they can output the results to the Command Window and a text file through a details option. The details of all the application features are outlined in chapter 3 and model tutorials are outlined in chapter 4.

2.1.1.2 Error checking on for 2D and 3D models

The application performs error checking by validating the Add, Modify, or Remove action, monitoring the various user inputs, and checking the method of joints requirement of three constraints for 2D models and six constraints for 3D models. In addition, for 2D models, the following equation must be met, $2 * \text{number of joints} = \text{number of members} + 3$. For 3D models, the equation is $3 * \text{number of joints} = \text{number of members} + 6$. Meeting the equation ensures the truss matrix is square, which is

required to take an inverse of the matrix. Adding the constraints to the proper axis makes certain the matrix is not singular. When a matrix is singular, the inverse of the matrix is unobtainable. For this reason, not all 3D trusses can be solved. The six constraints must be placed in a way that they restrain motion in all three directions and rotations. It is easier to assign constraints to 2D trusses because the user places two constraints on one joint and one constraint on another joint of the same plane. With 3D analyses, the user must think about the rotations of the model.

For complicated models, the users need to determine how to apply the constraints their particular model. If a constraint is not properly restraining against motion or rotation, then the reaction force will be zero. The lack of a resultant force means no force is being placed on the constraint. Hence, either it is not configured to restrain motion or no force is applied in that direction. As is the case of many 2D models, when all the loads are in the Y-axis, then the x constraints will have no reaction force. However, the x constraint still needs to be part of the model to ensure a square matrix and therefore a valid solution.

2.1.2 Projectile Motion

The projectile motion analysis package is divided into two applications; ramp and spring launched projectile and swing-launched projectiles. The individual applications calculate a projectile's launch energy to provide an initial velocity and direction for the projectile motion function. One can select to ignore drag, use a simple initial velocity

loss, or calculate the drag relative to a set of input coefficients. When the projectile reaches the ground, or a specified target, the projectile's path terminates.

2.1.2.1 Ramp and Spring Energy Dynamics Launch Velocity Calculation

The ramp and spring dynamics application utilizes a combination of initial kinetic and potential energy conditions. The user can choose to launch at the ramp angle or a specific launch angle. Friction and other energy losses for both the kinetic and potential energies account for energy loss as a percentage reduction.

The two general types of analysis performed by this application are shown in Figure 2.5. The first analysis is for a projectile with an initial velocity traveling up or down a ramp. The second analysis calculates the stored energy from a linear spring.

This application considers the change in kinetic energy from the ramp, then, if necessary, applies the launch energy. The ramp energy is defined by the initial kinetic energy and the change in height from the ramp,

$RampEnergy = KE_{INITIAL} + PE_{HEIGHT}$. The kinetic energy is

$$KE_{INITIAL} = \frac{1}{2} mass * Vmag^2, \text{ and the change in energy due to height is}$$

$PE_{HEIGHT} = mass * gravity * dy$. When the projectile is moving up the ramp, the entered dy value is positive, which causes a reduction in kinetic energy because gravity is negative. When the projectile moves down the ramp, the entered dy value is negative, which is multiplied by the negative gravity value and increases the kinetic

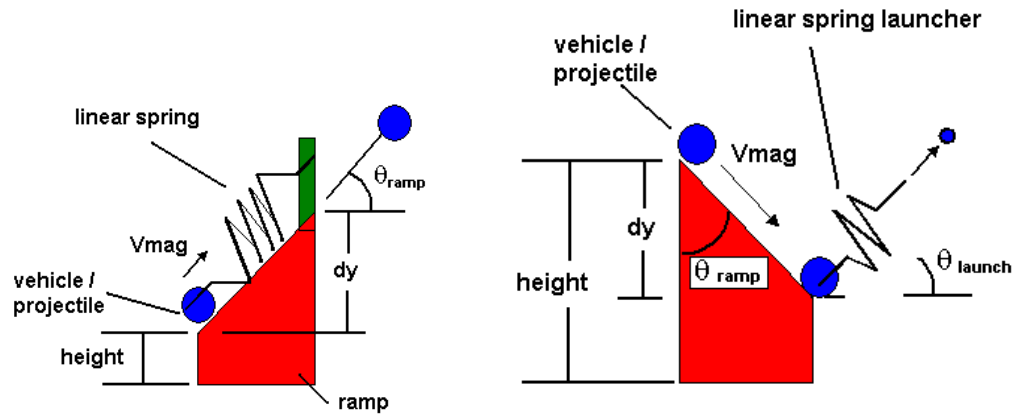


Figure 2.5 Ramp and Spring Energy Typical Analyses

energy. The user can specify the initial height, change in height, ramp angle, launch angle, vehicle's mass, and projectile's mass.

With a linear spring coefficient k and the initial and final spring displacement, the applied potential energy from the spring can be calculated,

$$PE_{SPRING} = \frac{1}{2} k_{Spring} * x_{Final}^2 - \frac{1}{2} k_{Spring} * x_{Initial}^2 .$$

The kinetic and potential energies can be added together,

$$Total \ Energy = RampEnergy + PE_{SPRING} .$$

The user can choose to model

only one of the energies. The launch velocity is determined by

$$V_{LAUNCH} = \sqrt{\frac{2 * Total \ Energy}{mass}} .$$

2.1.2.2 Swing Energy Dynamics Launch Velocity Calculation

The swing launched projectile uses a swinging mass with a given center position and radius of movement as shown in Figure 2.6. As the projectile swings from its starting angle of 95° to its release angle of 280° , the projectile has a gain in kinetic energy. At the release, the launch velocity is calculated from the resulting kinetic energy relative to the mass of the projectile using $KE = \frac{1}{2} mass * V_t^2 + mass * gravity * dy$. A vehicle starts with an initial velocity or drops from rest at the specified start angle. Similar to the ramp dynamics, a loss can be associated with each energy transfer. A projectile can be launched tangent to the release angle, or at an angle that is offset to the tangent release, or at a specified angle.

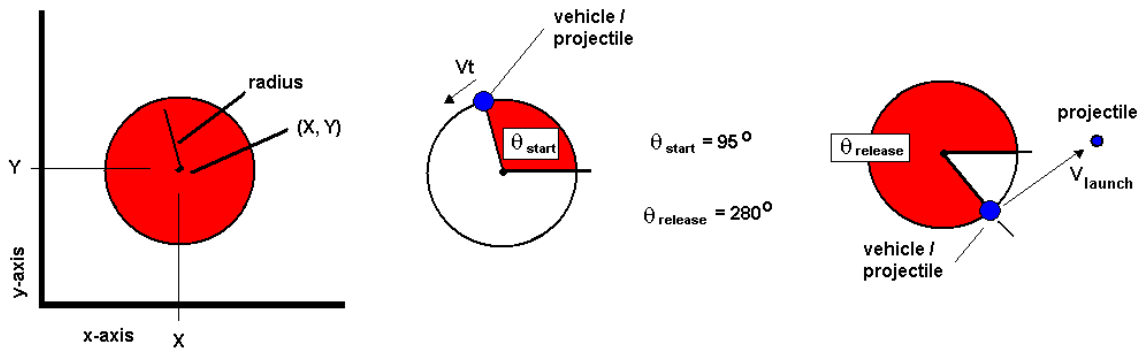


Figure 2.6 Swing Energy Description

2.1.2.3 Projectile Motion Calculations from Launch Velocity

Both dynamics applications can calculate a projectile's path by entering an initial velocity and launch angle. The launch velocity is either specified by the user or calculated by the transfer of energy from the ramp, spring, or swing.

The ability to calculate a projectile's path from an initial velocity and launch angle removes the limitation of the launching type. If a new method of launching is required for the team project that cannot be computed with the dynamics applications, the students can calculate the launch velocity for their design separately and then enter the launch information into either application to determine the flight path.

When neglecting drag, the launch velocity and angle are used to calculate the flight path using $x(t) = x_0 + V_{x_0} * t$ and $y(t) = y_0 + V_{y_0} * t + 0.5 * g * t^2$. The initial x and y positions, x_0 and y_0 , can be specified in each application. The initial velocities, V_{x_0} and V_{y_0} , are the x and y components of the launch velocity. For a specified time t , the equations predict the x and y position at that time.

Both applications can account for drag using one of two methods. For a simple drag method, the calculated launch velocity is reduced by a specified percentage. The simple method is intended to provide a quick initial answer before the complex drag method is used. The complex drag method uses a set of equations to determine the flight path with drag. The equations are $x(t) = x_0 + (V_{x_0} / \lambda) * (1 - e^{-\lambda t})$ and $y(t) = y_0 + (-g/\lambda^2 + V_{y_0} / \lambda) * (1 - e^{-\lambda t}) + (g / \lambda)t$. The initial x and y positions, x_0 and y_0 , are specified in the user inputs or calculated by the program. The initial velocities, V_{x_0} and V_{y_0} , are the x and y components of the previously calculated launch velocity. λ is the

Table 2.1 Simple and Complex Drag Calculations

Drag Method	Coefficients	Implementation
Simple	Drag Loss %	$\text{Velocity\_launch} = ((100 - \text{Drag Loss \%}) / 100) * \text{Velocity\_launch}$
Complex	k – Drag Coefficient	$\lambda = k / \text{mass of projectile}$ $g = \text{gravity}$ $x(t) = (V_{x_0} / \lambda) * (1 - e^{-\lambda t})$ $y(t) = (-g / \lambda^2 + V_{y_0} / \lambda) * (1 - e^{-\lambda t}) + (g / \lambda)t$

result of the entered coefficient of drag divided by the mass of the projectile. These simultaneous equations separately calculate the x and y positions for a specified time t . The complex drag option is outlined in the Boresci / Schmidt Example 14-7 [11] in chapter 4. The drag method calculations are shown in Table 2.1.

2.2.2.4 Modifying Plots

If the user wants to customize an output figure, then the plot can be put into a separate figure window. The axis, title, labels, legend, and any other plot formatting commands can be issued from the Matlab Command Window.

2.2 Deciding to use Matlab

All aspects of the applications can be programmed in any language. Neither the calculations nor the visual layout requires Matlab. The projectile motion function, in

particular, could have been programmed in an Excel Spreadsheet as outlined in Chapter 1. Matlab is the preferred software because of its combination of the matrix analysis, general plot displaying, and numeric analysis. All can be performed within Matlab, whereas Visual C++ (VC++) needs a complicated data storage process and an external matrix solver, like PETSc [12]. In addition, Matlab is the programming language taught in *engage*.

Matlab is designed to handle matrix computations similar to those needed for a truss analysis. The equation $[A]\{x\} = \{-b\}$ is an excellent example of Matlab's ability to solve matrices. This is a standard linear algebra equation. Matrix $[A]$ contains the coefficients that describe the mathematical relationships for the truss's joints and members, vector $\{-b\}$ is the known external forces, and vector $\{x\}$ is the unknown member and reaction forces. To solve this equation, the inverse of $[A]$ is needed, converting the equation to $\{x\} = [A]^{-1}\{-b\}$. In this form, the equation calculates the unknown member and reaction forces in vector $\{x\}$. In Matlab, this process is accomplished by the command $x = A \setminus -b$. Other languages are unable to build the matrix $[A]$, which easily makes them unable to take the inverse and solve. Clearly, Matlab is the best choice for the matrix computations needed to solve a complex truss matrix.

In Matlab, detailed plots can be created using a small number of commands, whereas in VC++, the user must issue commands to draw each individual characteristic of the plot, such as drawing the specific coordinates of the legend or axis labels. In Matlab, the legend's text and placement are simple commands to implement. The axis labels are automatically drawn centered and below the axis. In VC++, the user must

write code to calculate the label's position, and then draw the text at that location.

Additionally, the rotate feature in Matlab for the truss application is extremely simple to implement. In Matlab, the “rotate3d on” function turns a plot window into a mouse active three-dimensional window that enables free rotation of the view. To implement this action in VC++, one must load directx [13] or openGL [14] graphics.

Matlab simplifies the numerical analysis by having analysis functions built into the language. It is extremely code efficient to take a set of data, find a best-fit curve, and then perform whatever analysis is needed in Matlab. Once again, external code can be written to perform the math operations in VC++, but the functions are not built into the language.

2.3 Help Files

These applications are of little benefit if the students are unable to use the software. Each application includes a HTML based help file that can be opened from the embedded toolbar. The help file includes information about the various buttons and user entry areas, and how those options affect the analysis. Also, there are sample problems for the students to use as tutorials.

The combination of help files and sample walkthroughs reinforce the notion of the “random user”. When programming functions, the students are taught how to display help information associated with the function, and then taught how to make the help file for the “mysterious random user”. With these applications, they are that “random user”,

they learn the importance of the help files, making an intuitive interface, and writing organized and commented code, because they need to use each of the features to obtain an analysis that directly relates to their team project.

Chapter 3 - Application Features

The intention of this chapter is to discuss the capabilities of the individual features of the applications. Examples, verification, and tutorials are outlined in chapter 4.

3.1 Truss Analysis Application

The Truss Solver Application is capable of building and solving a wide range of 2D and 3D bridge-truss models. The application, shown in Figure 3.1, consists of nine specific features:

1. Toolbar Menu
2. Feature Menu
 - a) Joint
 - b) Member
 - c) Constraint
 - d) Force
3. Modify Menu
 - a) Add
 - b) Modify
 - c) Remove
4. Enter Values Menu
5. Analysis Menu
6. Axis Menu

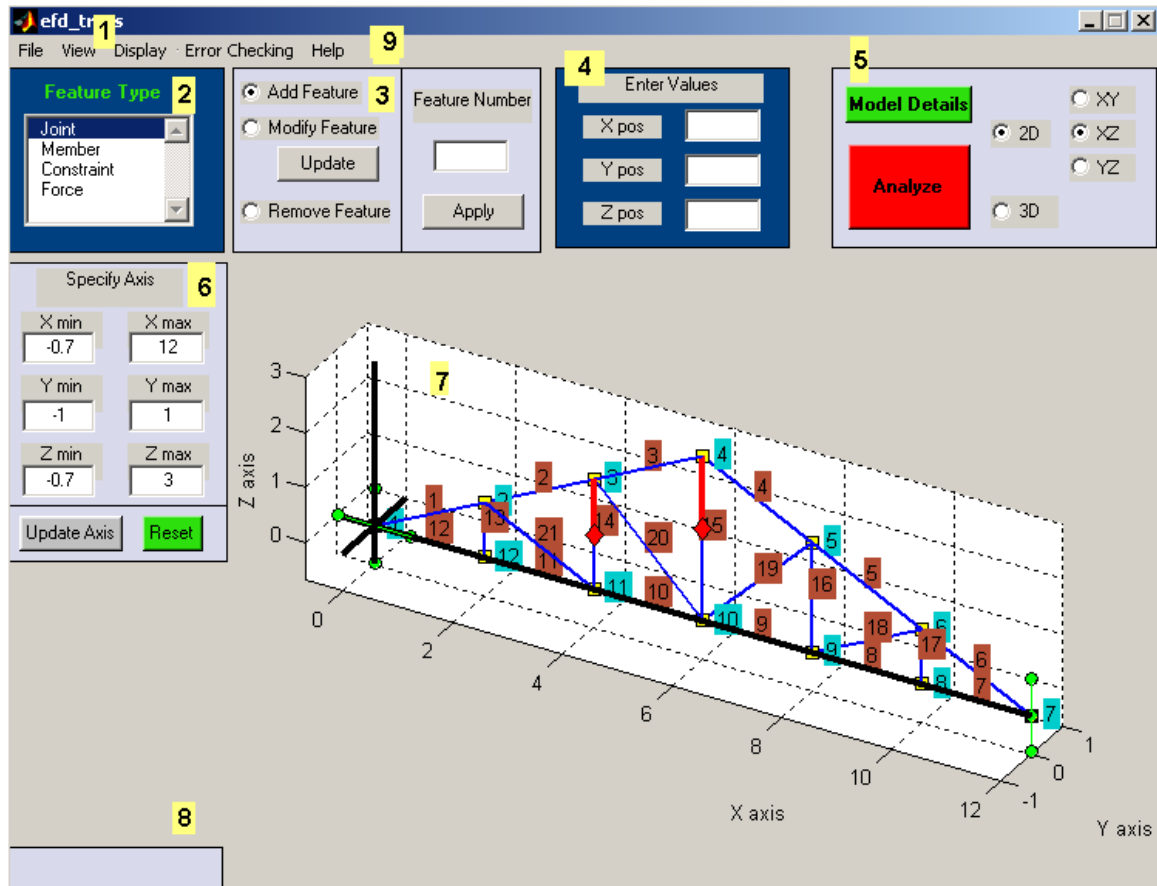


Figure 3.1 Engage Truss Solver

- 7. Draw Area Menu
- 8. View Menu
- 9. Help File

The application features needed to build and analyze the truss are easily accessible to the user. Every user input feature is shown on the main user area. The layout is important, because the students do not need to hunt for the particular user entry menu. Each feature contributes to the complete application.

Figure 3.2 Truss Solver Toolbar

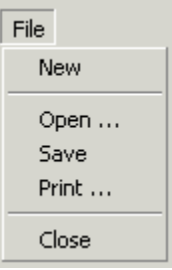
	File: <i>New</i> - Clears the current truss model <i>Open</i> - Opens a *.mat file (must have been created by EFD_TRUSS) <i>Save</i> - Saves a *.mat file, every save is a "Save As" <i>Print</i> - Prints Display <i>Close</i> - Closes EFD_TRUSS

Figure 3.3 Truss Solver, File Menu

3.1.1 Toolbar Menu

The Toolbar Menu, shown in Figure 3.2, contains all the file management and user option controls. The Toolbar consists of five headers, File, View, Display, Error Checking, and Help.

All the file management is inside the *File* header, shown in Figure 3.3. The user can reset the model, open an existing model, save the current model, print the figure, and close the application.

The *View* header, shown in Figure 3.4, allows the user to show a particular view. One can choose the Top, Right, Front, and Isometric views.

The *Display* header, shown in Figure 3.5, controls what features are displayed. From this header, the user can show or hide the joint numbers, member numbers, constraints, forces, axes, axis labels, and grid. Also, there is a toggle feature. If a

	View: <i>Top (XY)</i> – Sets view to XY plane <i>Front (XZ)</i> – Sets view to XZ plane <i>Side (YZ)</i> – Sets view to YZ plane <i>Iso</i> – Sets to an isometric view

Figure 3.4 Truss Solver, View Menu

	Display: <i>Toggle Display</i> - Toggles between displaying all options and displaying nothing. The "Axis Labels" and "Grid" are not modified. <i>Joint Numbers</i> - Show (checked), Hide (unchecked) <i>Members Numbers</i> - Show (checked), Hide (unchecked) <i>Constraints</i> - Show (checked), Hide (unchecked), if the truss has been analyzed, this option will toggle the reaction forces. <i>Force</i> - Show (checked), Hide (unchecked) <i>Axes</i> - Show (checked), Hide (unchecked) <i>Grid</i> - Show (checked), Hide (unchecked)

Figure 3.5 Truss Solver, Display Menu

majority of items are checked, all will become unchecked. The converse is also true. The joints and member connections are always shown. The grid and axis labels are unaffected by the toggle.

The error checking, shown in Figure 3.6, verifies some of the basic requirements for method of joints analysis and recovers damaged models. The Verify Number of Constraints option verifies that the number of constraints is the proper number for the given analysis, three constraints for 2D, six constraints for 3D. For 2D models, Verify 2D Plane verifies the analysis plane is the same as the model's plane. The Model Recovery feature removes any member, constraint, or force that is attached to a joint that does not exist. The internal error checking should ensure that a feature is not connected to a non-existent joint. However, if the user finds a way to bypass the error checking, then it will corrupt the model file by trying to draw a feature for a joint that does not exist. The Model Recovery feature scans and removes the member, constraint, and force variables attached to the nonexistent joint.

The help header, shown in Figure 3.7, includes two options, the help guide and an about pop-up. The help guide contains information about each feature and sample walkthroughs. The about pop-up gives information about the title of the application, it's programmer, etc. Detailed examples of the Quick Guide are included in chapter 4.

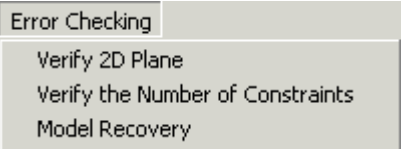
	Error Checking: <i>Verify 2D Plane</i> - Compares the model to the selected plane. If a joint or constraint exists on an illegal plane, the user will be notified. <i>Verify the Number of Constraints</i> - If the number of constraints does not correspond to the 2D / 3D analysis, the user will be notified. <i>Model Recovery</i> - This is a safety valve. This option will remove any member, constraint, or force that references a joint that does not exist. The user will only need this option if the model continuously fails to draw.

Figure 3.6 Truss Solver, Error Checking Menu

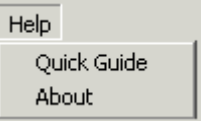
	Help: Quick Guide - General Help About - Program Information

Figure 3.7 Truss Solver, Help Menu

3.1.2 Feature Menu

The Feature Menu, shown in Figure 3.8, is used to add new features for analysis. The user may create joints, members, constraints, and forces. This menu works hand in hand with the *Modify* and *Enter Values* menus. Once the user selects a feature (joint, member, constraint, or force), he must provide information to the Modify Menu's Add, Modify, or Remove, then based on the operation and feature enter a Feature Number, and finally, enter the feature data into the Enter Values Menu. The Feature Menu, Modify Menu, and Enter Values Menu are shown in Figure 3.9 set to the Joint Feature, therefore the Enter Values menu shows *X pos*, *Y pos*, and *Z pos*.

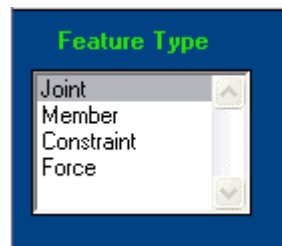


Figure 3.8 Truss Solver, Feature Menu

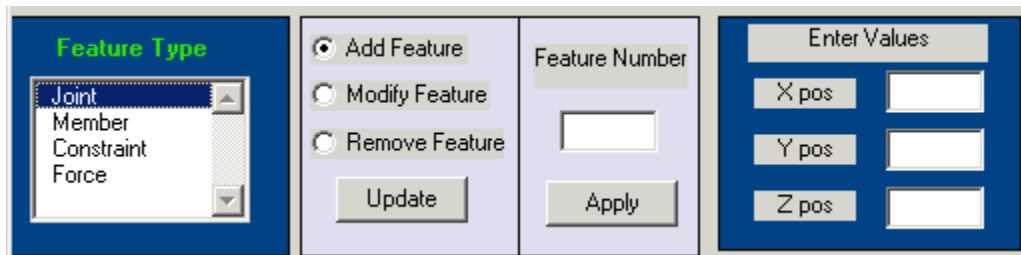


Figure 3.9 Feature, Modify, and Enter Values Menus

The Feature Menu controls the inputs of the Enter Values Menu. Depending on the feature type, the enter values menu corresponds to the feature selection. For instance, when the feature menu is set to *Joint*, the enter values menu shows inputs for the X, Y, and Z positions. When the feature menu is set to *Force*, the enter values menu shows input labels for the forces in the X, Y, and Z direction (“FX”, “FY”, and “FZ”). The Modify Menu and Enter Values Menu are discussed later in this chapter, sections 3.1.3 and 3.1.4, respectively.

Obviously, one cannot create a member unless the joints have first been created. When one attempts to create a member, two joints must be entered. Internal error checking verifies the user entries are valid.

Joints and members are numbered with respect to the order in which they are created. Constraints and forces are numbered with respect to the joint that they are attached. Joint, constraint, and force values are based on X, Y, & Z components. Members are based on the two joint numbers.

3.1.3 Modify Menu

The Modify Menu, shown in Figure 3.10, is used to add, modify, or remove a feature. A specific feature is modified or removed by entering its *Feature Number* and pressing apply to make the changes that have been selected. Obviously, error checking ensures the user cannot modify or remove a feature that does not exist and he cannot create the same type of feature with the same feature number values as an existing feature. For instance, if the user adds two joints with XYZ coordinates of (0,0,0), or adds

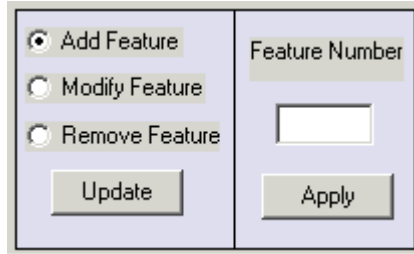


Figure 3.10 Truss Solver, Modify Menu

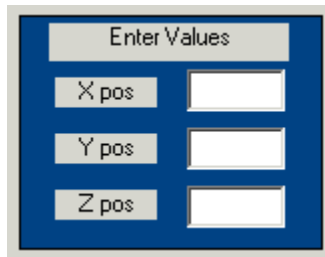


Figure 3.11 Truss Solver, Enter Values Menu

two separate force entries to joint 5, then the application reports these errors the moment they occur.

3.1.4 Enter Values Menu

The Enter Values Menu, shown in Figure 3.11, is used to enter X, Y, & Z data for Joints, Constraints, and Forces. For Members, joint numbers are entered here. This menu works with The *Modify Menu* and *Feature Menu*.

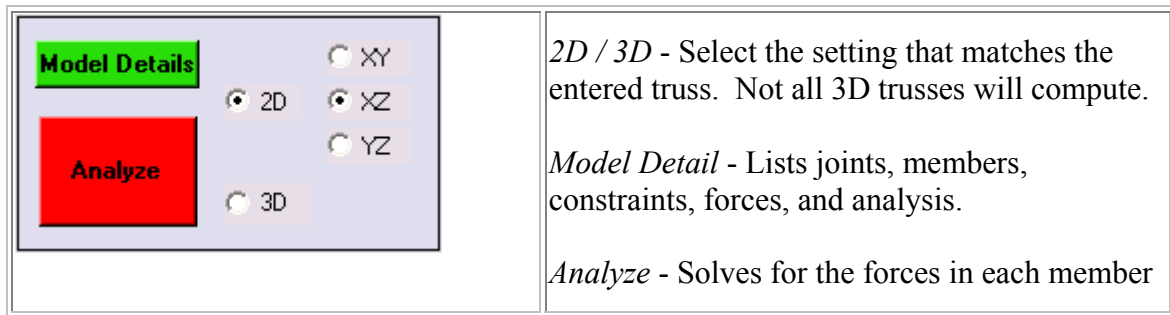


Figure 3.12 Truss Solver Analysis Menu

3.1.5 Analysis Menu

The Analysis menu is shown in Figure 3.12. The *Analyze* button will take the existing model and build a coefficient matrix and a force vector. After the matrix and vector data has been entered, Matlab solves for the member and reaction forces. The member and reaction forces are then displayed on the model.

The Model Details button lists all the model information, such as the joints, members, constraints, and forces. When clicked, a data entry box, shown in Figure 3.13, will receive specific model information from the user. This information is shown at the top of the output. If the model has been analyzed, the option will output a data file displaying the member and reaction forces to the Command Window, as well.

The 2D / 3D selection is critical in determining the correct analysis. In 2D, if the model's plane and analysis' plane do not correspond, the incorrect rows and columns will be removed from the geometry matrix. In this case, the results will usually include an infinite member force. Because the users have a general background of the method of joints technique from the *engage* curriculum, they will know some type of error has occurred. If a user cannot find the error, the provided error checking can be used to

Model Details

Model / Project Name:
Project 1

Your Name:
not Rooney

Team:
The Nuclear Orange Cheeto Fingers

Description:
Structural Analysis of Project 1

Units:
English

OK Cancel

Figure 3.13 Truss Solver, Model Details Menu

detect the problem.. In this case, the Toolbar option *Verify 2D Plan*” in *Error Checking* will inform the user of the cause for the incorrect analysis.

3.1.6 Axis Menu

The Axis Menu, shown in Figure 3.14, allows the user to set the axis limits that scale the plot window. The user can use this menu to zoom in or out of the model. The user enters a minimum and maximum x, y, & z values and hits *Update Axis*. The drawing area will clear and the new axis coordinates will be applied on the plot. This menu also allows the user to reset the view and analysis results by hitting the *Reset* button.

Specify Axis

X min	X max
-3.6	75.6
Y min	Y max
-3.6	9
Z min	Z max
-3.6	10

Update Axis Reset

Figure 3.14 Truss Solver, Axis Menu

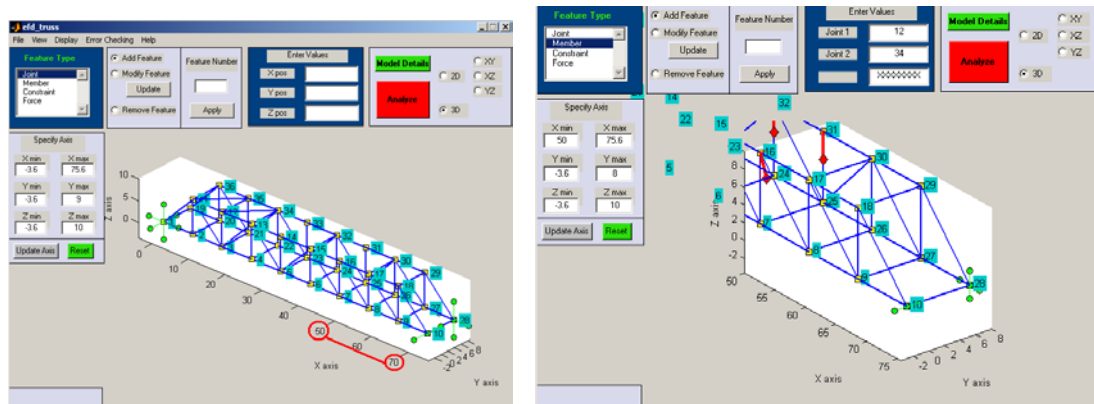


Figure 3.15 Full Truss (left) and Zoomed View (right)

If the user needs to examine a particular portion of the model, then the axis limits can be modified to any numeric value required by the user. In this case, the X_{min} value has been changed from -3.6 to 50 , (see Figure 3.15).

As shown in the zoomed view, Matlab draws some text labels and lines outside of the plot window. Fortunately, they are drawn beneath the menus. While the loose text labels and lines are a nuisance, they do not interfere with the appearance of any user input menus.

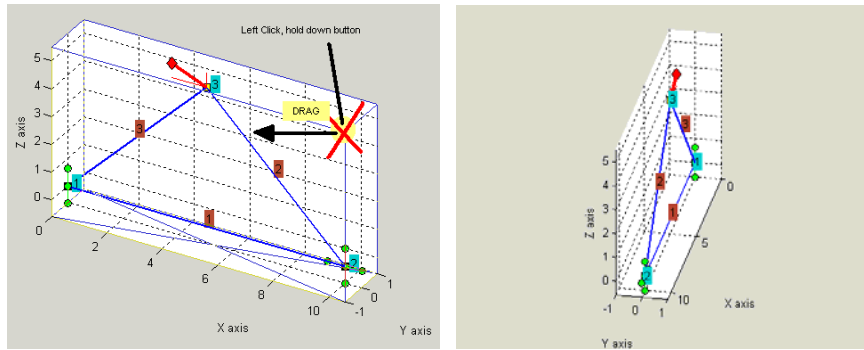


Figure 3.16 Truss Solver, Draw Area

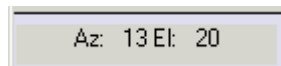


Figure 3.17 Truss Solver, View Menu Output

3.1.7 Draw Area

The Draw Area is the visual output of the entered truss data. While this view can be adjusted using the *Display* menu, the user can also rotate the view to any orientation. When the user holds down the left mouse button on the picture (white area) and moves the mouse, then the view will rotate, shown in Figure 3.16.

3.1.8 View Display Menu

The View Display Menu, shown in Figure 3.17, only appears while the view is being rotated. It is possible to hide the display, but the user may want a certain customized view. Therefore, the user can set the Azimuth and Elevation to the values

that were used previously in the report or presentation. So, each picture is from the same perspective.

3.1.9 Help File

The help file contains two main sections, feature information and walkthrough tutorials. The user can access the help file by clicking on the *Help* item on the toolbar, and then selecting *User Guide*. Once clicked, a web page help file appears. The help file's homepage is shown in Figure 3.18. The user can find feature menu information by clicking on the list on the left, or by clicking on the picture of a particular menu.

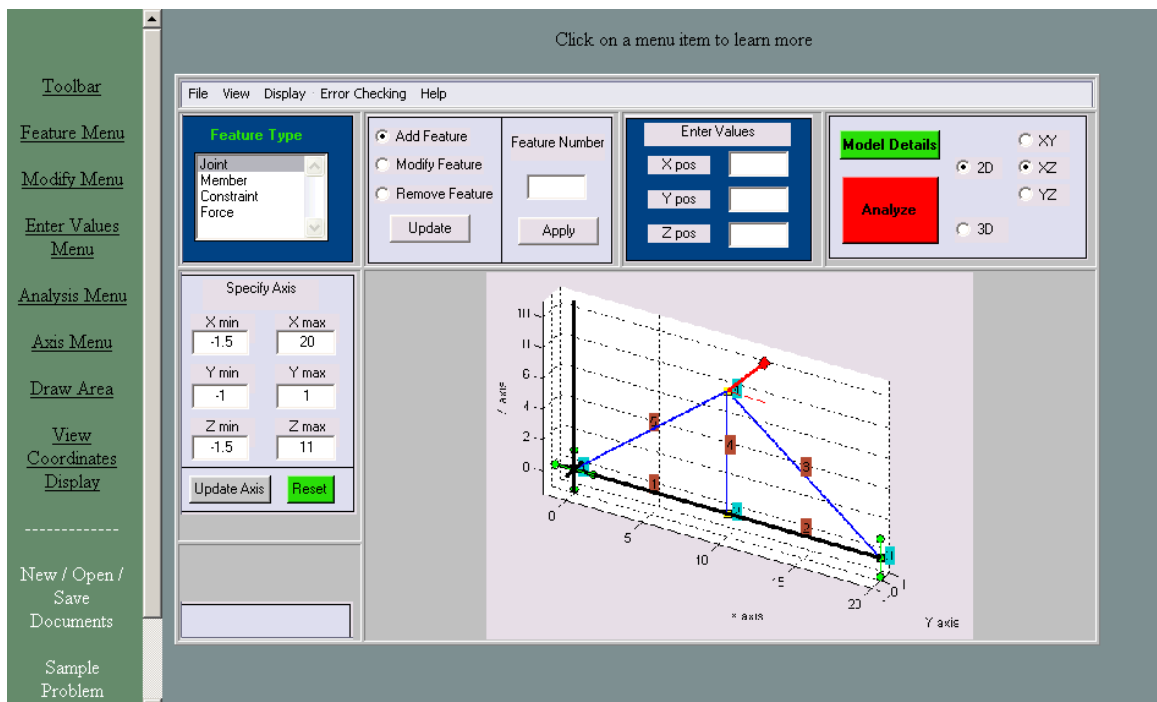


Figure 3.18 Truss Solver, Help File

The feature information is similar to the information shown in sections 3.1.1 through 3.1.8. If the users have a question about a particular menu, then they can open the help file and click on the topic of choice.

The sample problems contain a few walkthrough tutorials. The walkthrough tutorials help the users build their first model and provide sample problems to showcase the various types of analysis. The tutorials are described in chapter 4 in detail.

3.1.10 Internal Error Checking

The application constantly monitors the user input. The three main types of input errors are non-numeric values, duplicate entries, and attaching information to joints that don't exist. If the user makes one of these errors, The application will provide feedback to the user.

Figure 3.19 shows an input error. The user entered a non-numeric value. All entries must be numbers.

Figure 3.20 shows a user attempting to make a duplicate entry. This error checking works for joints, members, constraints, and forces.

Figure 3.21 shows the user attempting to apply constraint information to a joint that does not exist. This error checking also works for forces and members.

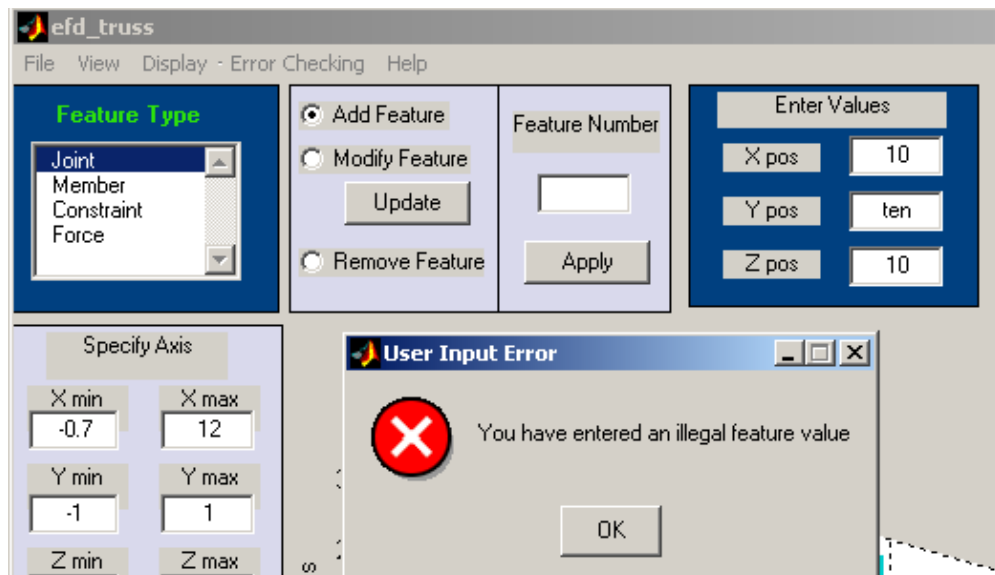


Figure 3.19 Truss Solver, Non-numeric Value Error

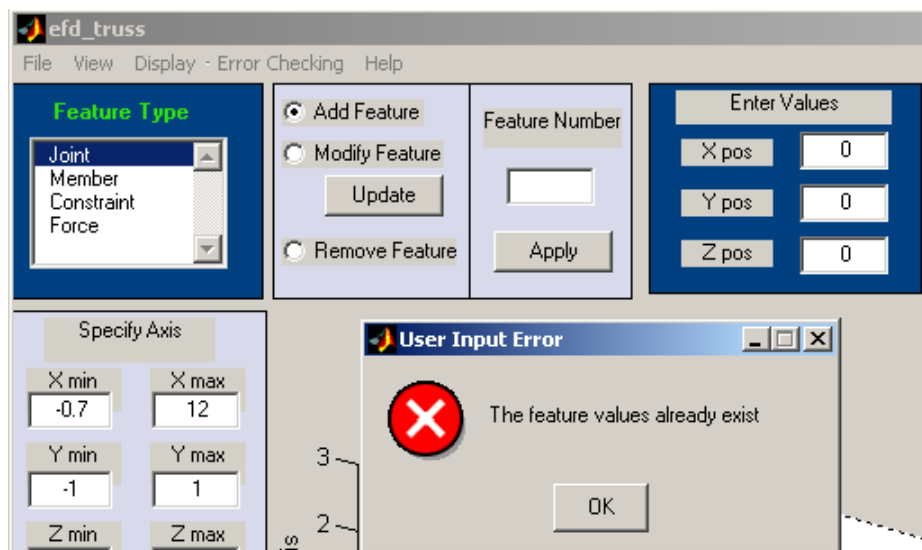


Figure 3.20 Truss Solver, Duplicate Entry Error

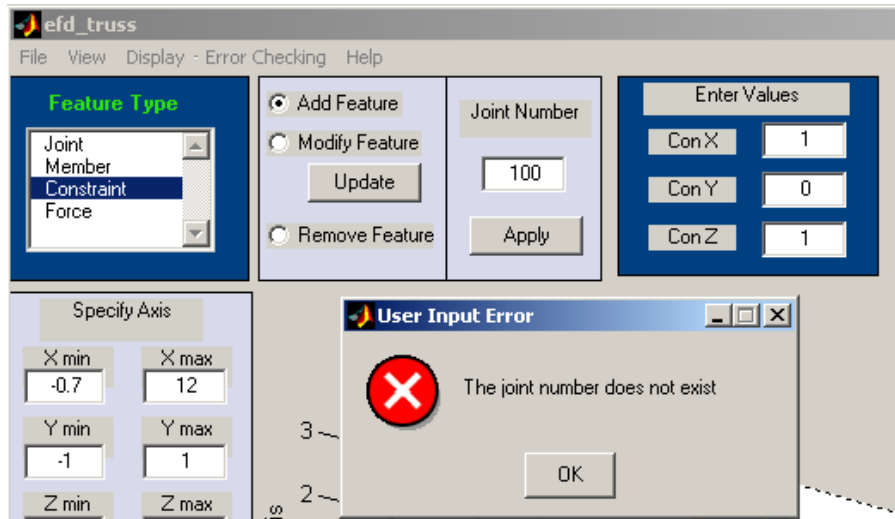


Figure 3.21 Truss Solver, Non-existing Entry Error

3.2 Ramp Launched Projectile Motion Application

The Ramp Launched Projectile Motion Application, shown in Figure 3.22, solves a wide range of energy transfer-projectile motion problems. The analysis is built from information from the following ten menus on the main draw area.

1. Toolbar
 - a. Display
 - b. Units
 - c. Help
2. Time Increment Menu
3. Ramp Energy Menu
 - a. Sliding down the ramp

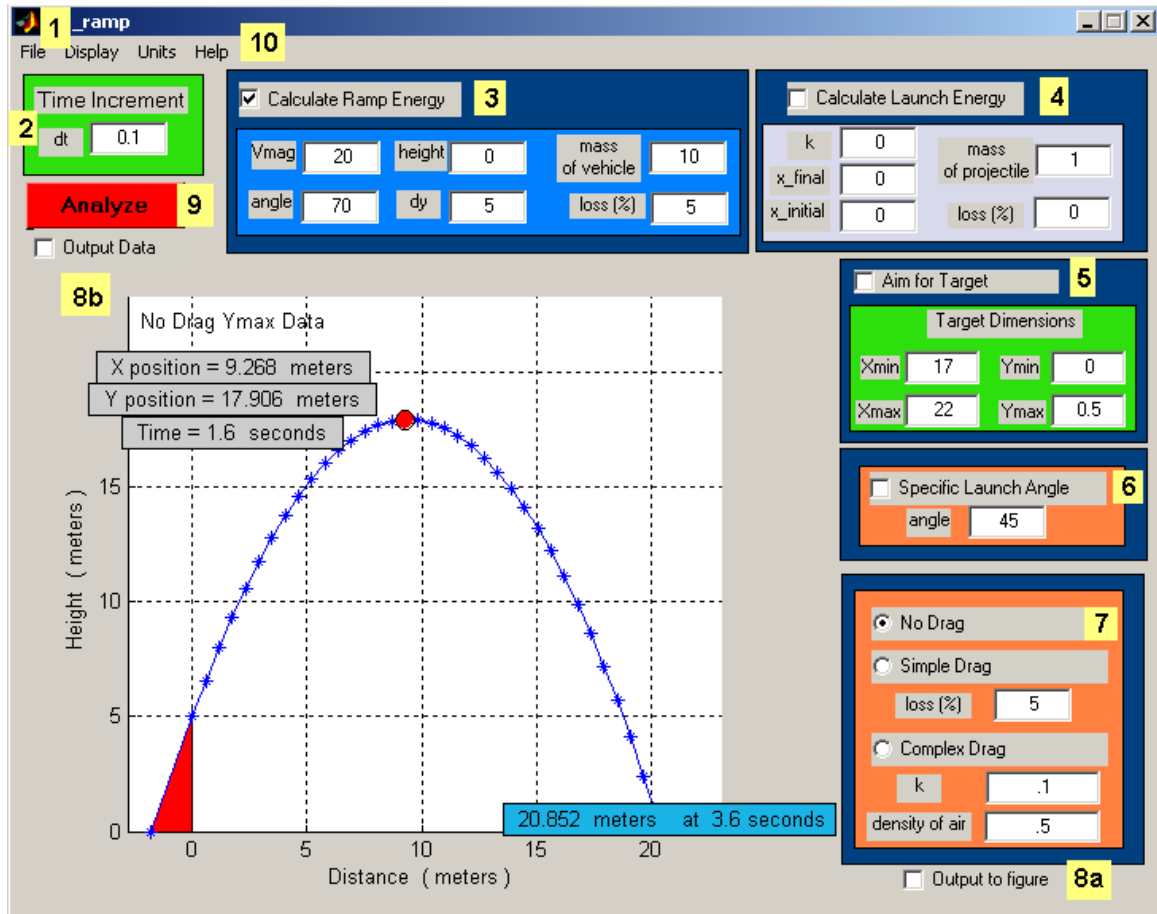


Figure 3.22 Ramp and Spring Launching Application

- b. Traveling up the ramp
- c. Generic launch (initial velocity and angle)
4. Launch Energy Menu
 - a. Launch energy with ramp energy
 - b. Launch energy and a specified angle
5. Aim for Target Menu
6. Specify launch angle Menu

Figure 3.23 Ramp and Spring Launching Application, Toolbar

7. Drag Menu
 - a. No Drag
 - b. Simple Drag
 - c. Complex Drag (calculation from drag coefficients)
8. Outputting Figures and Data
 - a. Main draw area
 - b. To a separate figure
 - c. Output data plots and results file
9. Analysis Button
10. Help File

3.2.1 Toolbar

The Toolbar Menu, shown in Figure 3.23, contains all the file management and user option controls. The Toolbar consists of four headers, File, Display, Units, and Help. The file menu is only to close the application, or print the figure. There is not a need to save these models, because the user entry is limited to a few pieces of data.

The display header, shown in Figure 3.24, controls the output to the screen, shown in Figure 3.25. It is possible to toggle each path on and off as well as the onscreen results.

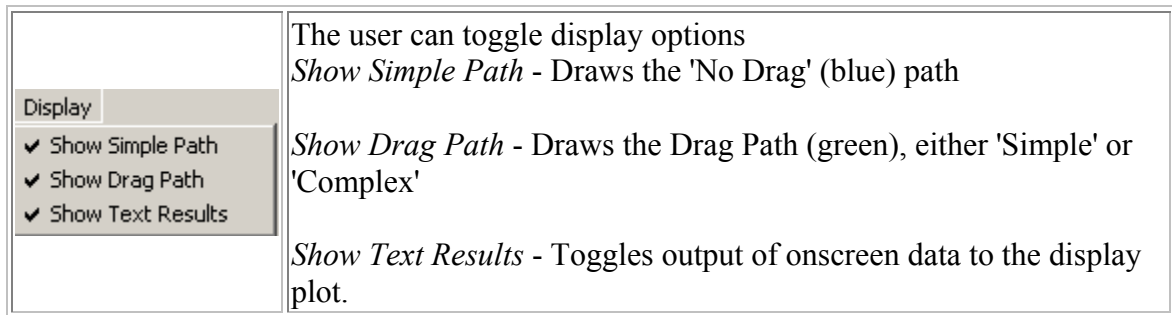


Figure 3.24 Ramp and Spring Launching Application, Display Menu

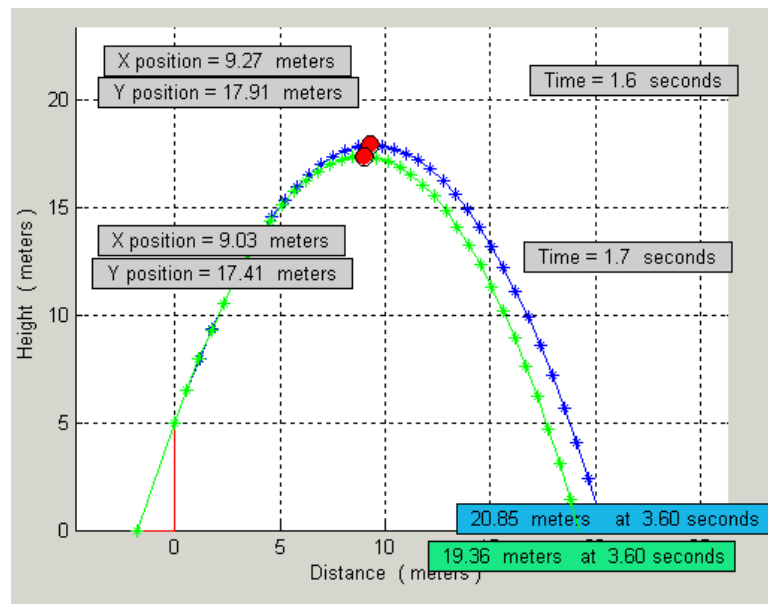


Figure 3.25 Ramp and Spring Launching Application, Sample Output

The Units header, shown in Figure 3.26, informs the application which units to use for the entries. When the user selects a unit system, a separate figure appears, shown in Figure 3.27. The figure lists the units for each user entry and remains open until the user closes the figure window. Optimally, the units would be next to each user entry, but there is not enough room for all the menus and the units.

3.2.2 Time Increment Menu

The Time Increment Menu controls the time steps for each of the iterations. Figures 3.28 and 3.29 are with the time increment set to 0.5 and 0.1 seconds. The process of selecting a time increment is outlined in chapter 4.

3.2.3 Ramp Energy Menu

This menu can simulate a vehicle moving up or down the ramp as explained in Chapter 2. Also, when properly configured, a user can specify launch velocity and launch angle.

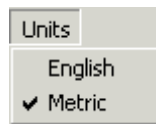


Figure 3.26 Ramp and Spring Launching Application, Unit Selection Menu

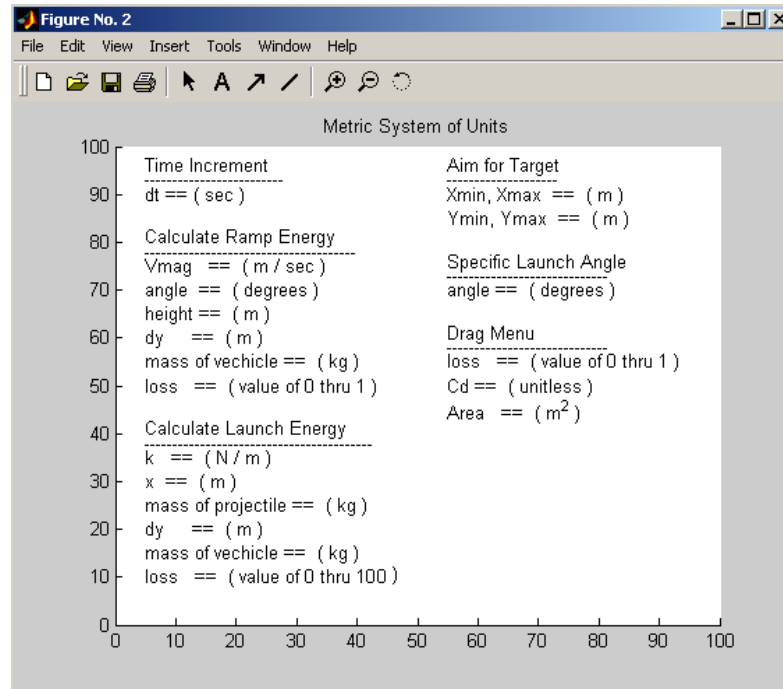


Figure 3.27 Ramp and Spring Launching Application, Unit Display

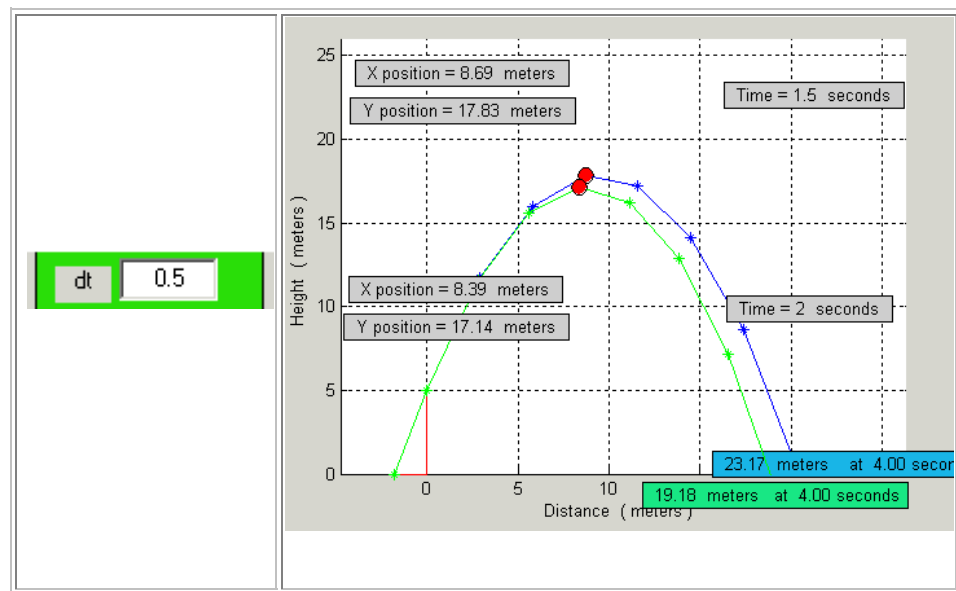


Figure 3.28 Ramp and Spring Launching Application, 0.5s Time Increment

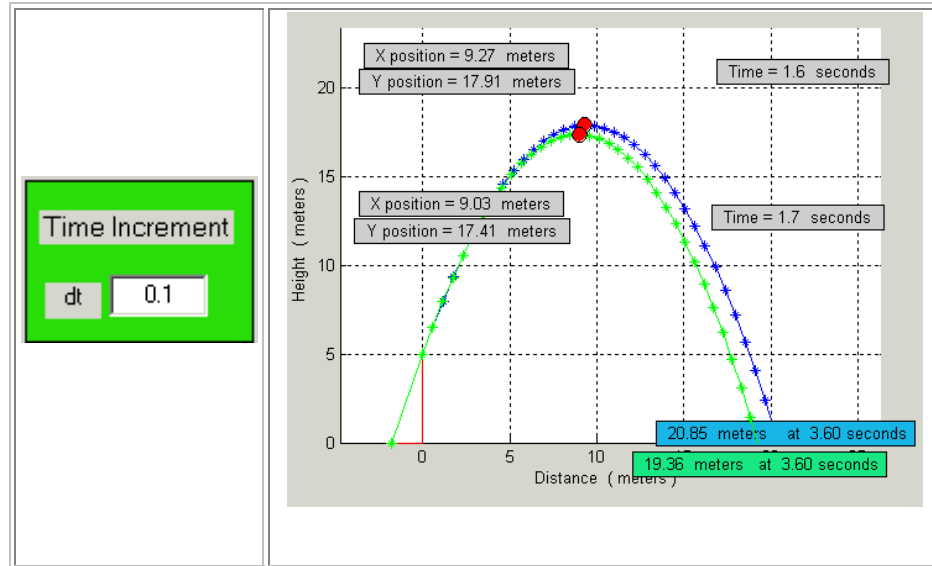


Figure 3.29 Ramp and Spring Launching Application, 0.1s Time Increment

3.2.3.1 Angle calculations

The user must input the *angle* and *dy* values to specify whether the projectile is moving up or down the ramp. To simulate the projectile moving up the ramp, the angle value must be between 0 and 180 degrees CCW from the x-axis and the *dy* value must be greater than zero. Figure 3.30 shows the upward ramp angles of 15, 40, 75, 105, 140, and 165 degrees CCW from the x-axis. To simulate the projectile moving down a ramp, the angle value must be between 181 and 360 degrees CCW from the x-axis and the *dy* value must be negative because the projectile is moving down relative to the positive y direction. Figure 3.31 shows the downward ramp angles of 195, 220, 255, 285, 320, and 345 degrees CCW from the x-axis. If the *dy* value is set to zero, then the ramp has no effect on the projectile's kinetic energy. An error message informs the user if the *angle*

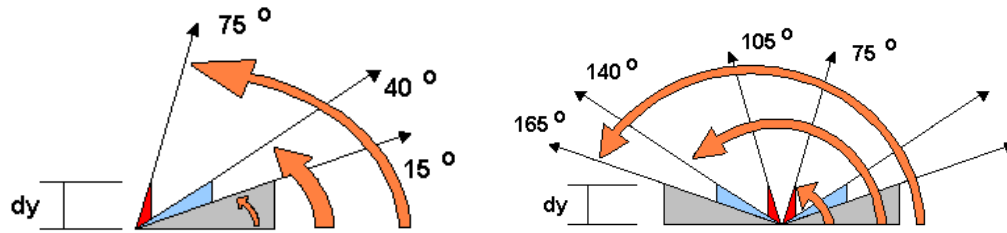


Figure 3.30 An upward ramp and positive dy value

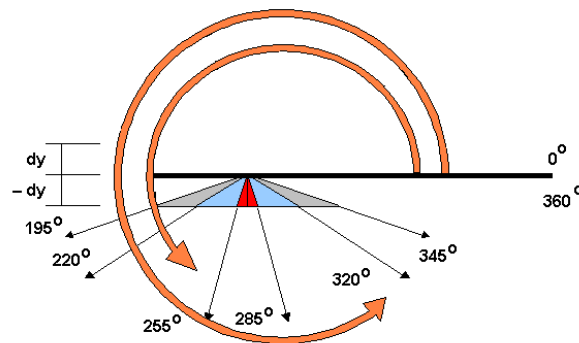


Figure 3.31 A downward ramp and negative dy value

and dy values do not correspond to a definite upward or downward ramp. For instance, a ramp angle of 60 degrees with a dy value of -5 feet is incorrect because the ramp angle of 60 degrees is associated with an upward ramp and the dy value of -5 feet is associated with a downward ramp.

3.2.3.2 Initial velocity up a ramp

Figure 3.32 shows the coefficient's values needed to simulate a vehicle moving up a ramp and launching a projectile at the same angle as the ramp. There are three

Calculate Ramp Energy					
Vmag	20	height	0	mass of vehicle	10
angle	70	dy	5	loss (%)	0

Vmag - Initial Velocity

angle - Ramp angle, must be between 0 and 180 to simulate upward ramp.

height - Initial height, must be greater than 0.

dy - Ramp height, must be greater than 0 to simulate upward ramp.

mass of vehicle - in kg or slugs

loss (%) - Percentage of energy loss during transfer, enter 0 to 100.

Figure 3.32 Ramp and Spring Launching Application, Ramp Upward

conditions. First, the *angle* value must be between 0 and 180 degrees. Second, the *dy* value must be greater than zero. Finally, the velocity's magnitude must be able to overcome the change in height. There is an error message verifying the velocity and height relationship. However, if the user enters a loss value from the Ramp Energy Menu, then the loss is taken from the energy remaining at the top of the ramp. The results of the user settings are shown in Figure 3.33.

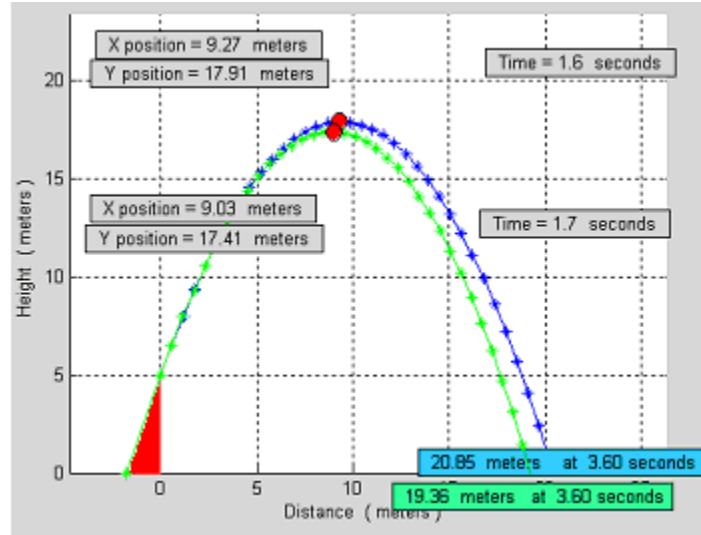


Figure 3.33 Ramp and Spring Launching Application, Ramp Upward Results

3.2.3.3 Initial velocity down a ramp

Figure 3.34 shows one set of coefficient values needed to simulate a vehicle moving down a ramp and releasing at that angle. There are two conditions. First, the *angle* value must be between 181 and 360 degrees. Second, the *dy* value must be less than zero. An error message informs the user if the *angle* and *dy* values do not correspond. The user specifies the initial velocity *Vmag*, ramp angle, initial height, change in height, mass of the vehicle, and loss due to the energy transfer. The result of the user data for the vehicle sliding down the ramp is shown in Figure 3.35.

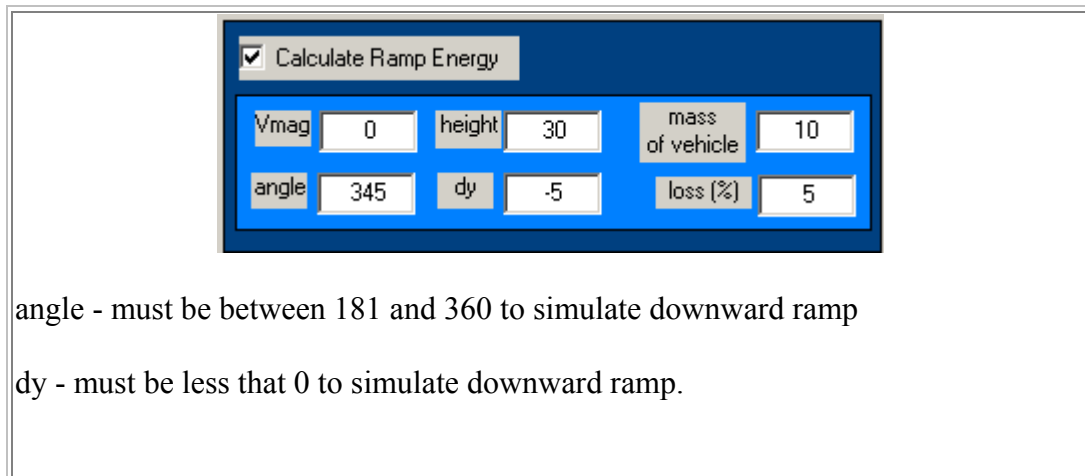


Figure 3.34 Ramp and Spring Launching Application, Ramp Downward

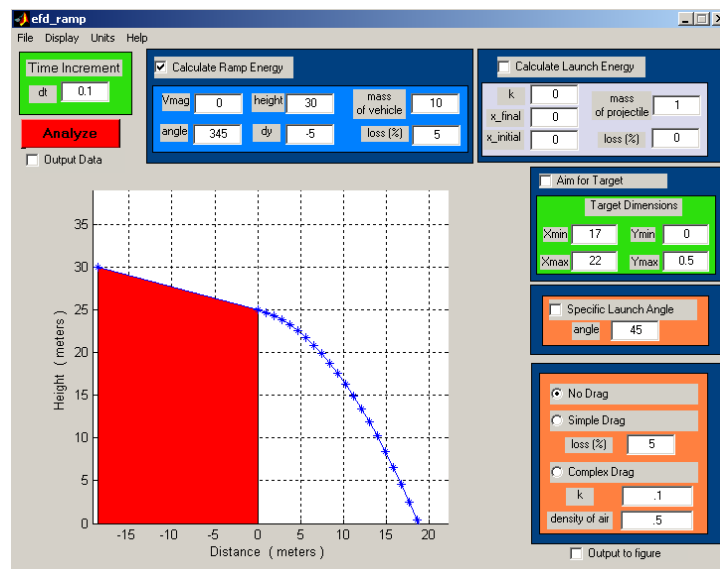


Figure 3.35 Ramp and Spring Launching Application, Specified angle (left) and Ramp Downward Results (right)

3.2.3.4 Simple Initial Velocity Launch

Figure 3.36 shows the coefficients necessary to calculate the projectile's path without any ramp or spring energy. If the dy value is set to zero, the calculation has no ramp energy effects, as in Figure 3.37. The velocity's initial vector is set by the *angle* value, unless an angle is specified.

3.2.4 Launch Energy Menu

The Launch Energy Menu, shown in Figure 3.38, is used to simulate the release of stored energy from a linear spring. The user enters the linear spring constant k , the initial and final stretch lengths ($x_{initial}$ and x_{final}) of the spring, the mass of the projectile, and the percentage of energy loss due to the transfer of energy. It is possible to use launch energy with or without the ramp energy. Figure 3.39 shows a vehicle sliding down a ramp, and then releasing spring energy at a specified angle. Figure 3.40 shows a stationary release of spring energy at a specified angle. The Specified Launch Angle Menu is discussed in Section 3.2.6.

Figure 3.36 Ramp and Spring Launching Application, Simple Velocity Analysis

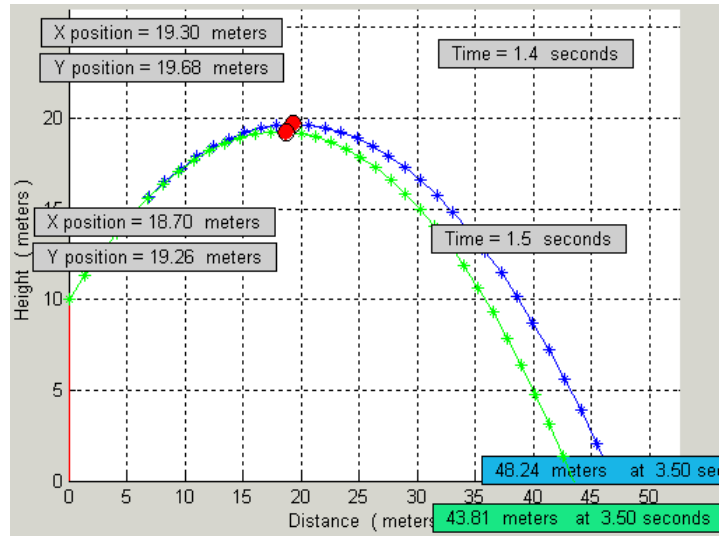


Figure 3.37 Ramp and Spring Launching Application, Simple Velocity Results

☒ Calculate Launch Energy		k - Spring Constant x_{final} - Final Spring Deflection $x_{initial}$ - Initial Spring Deflection $mass\ of\ projectile$ - In kg or slug. Only used when Calculate Launch Energy is checked. $loss(\%)$ - Percentage of energy loss during the transfer, 0 to 100.
k	10	
x_{final}	1	$mass\ of\ projectile$
$x_{initial}$	5	1
	$loss(\%)$	5

Figure 3.38 Ramp and Spring Launching Application, Launch Energy Menu

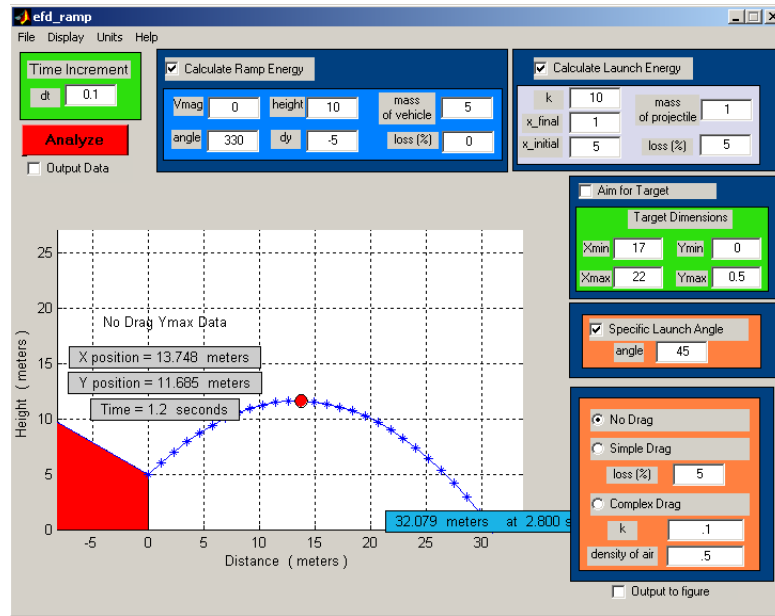


Figure 3.39 Ramp and Spring Launching Application, Combination Launch

Output

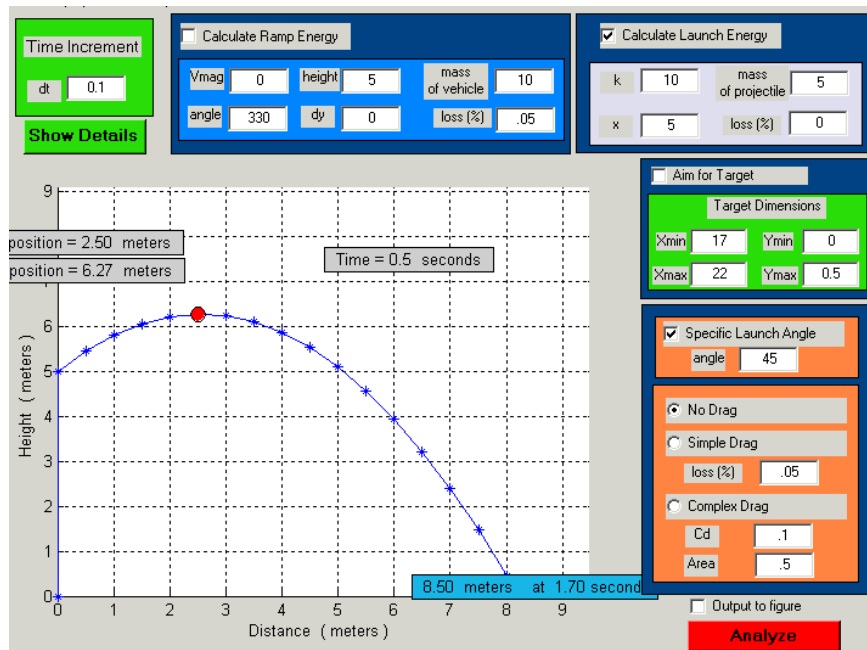


Figure 3.40 Ramp and Spring Launching Application, Spring Launch Output

3.2.5 Aim for Target Menu

The aim for target menu, shown in Figure 3.41, draws a rectangular target for the projectile. The user specifies a rectangular target using the values X_{min} , X_{max} , Y_{min} , and Y_{max} . When the projectile's path moves inside the target, the path terminates. If the path never moves inside the target, the path terminates when the project hits the ground at $y = 0$. Figure 3.42 shows the results with and without aiming for a target.

3.2.6 Specify Launch Angle Menu

This feature, shown in Figure 3.43, controls the use of a specified angle. When analyzing the spring energy, the user must specify the angle to launch the projectile. Also, it's possible the user will want to have a launch angle different from the ramp angle. In either case, to specify an angle, simply click on the checkbox and enter a value. Figure 3.44 shows the difference between a specified angle and a ramp angle using the same coefficients.

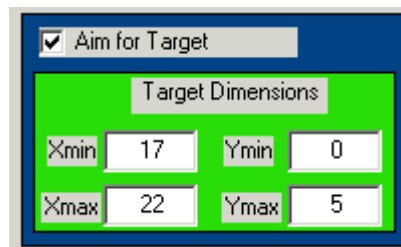


Figure 3.41 Ramp and Spring Launching Application, Target Menu

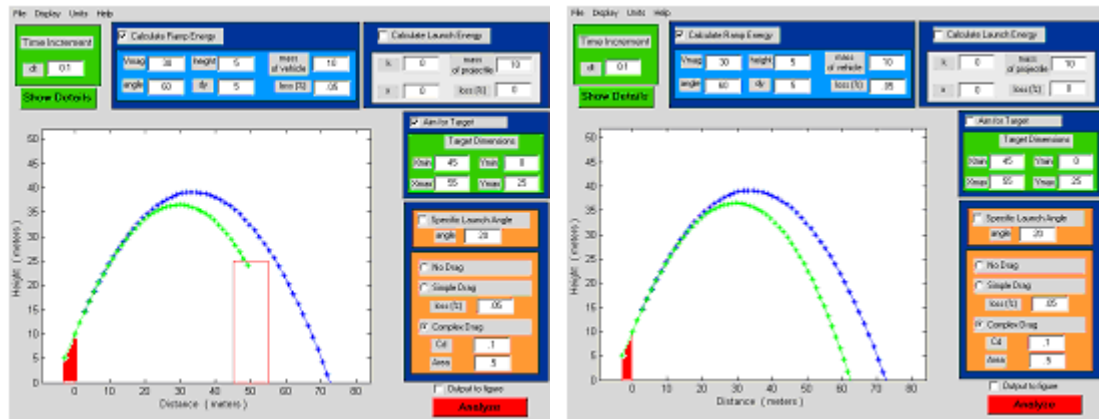


Figure 3.42 Ramp and Spring Launching Application, With (left) and Without (right) Target Menu

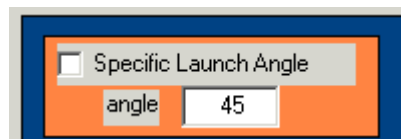


Figure 3.43 Ramp and Spring Launching Application, Specific Launch Angle Menu

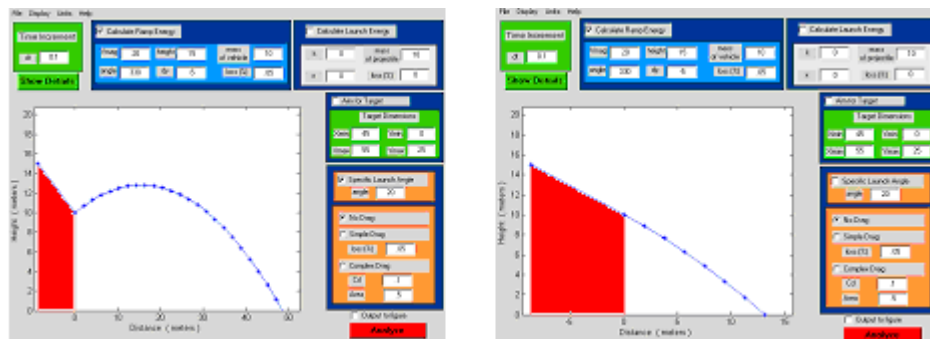


Figure 3.44 Ramp and Spring Launching Application, With (left) and Without (right) Specific Launch Angle

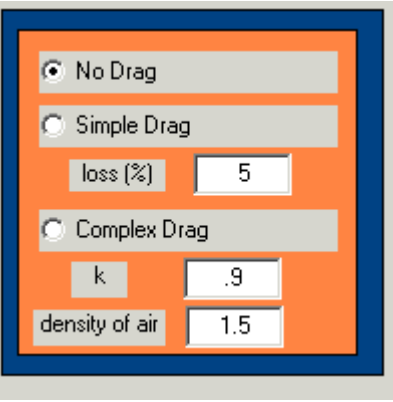
	<i>No Drag</i> - Drag is not considered <i>Simple Drag</i> - Reduces the launch velocity by a percentage. <i>Complex Drag</i> - Calculates drag with respect to the coefficients and the instantaneous velocity.

Figure 3.45 Ramp and Spring Launching Application, Drag Menu

3.2.7 Drag Menu

The Drag Menu, shown in Figure 3.45, is for the user to select the type of loss associated with drag. There are three options, ignoring drag, a simple drag loss calculation, and a coefficient of drag calculation. The “Simple Drag” option is a percent reduction in the initial launch velocity. The “Complex Drag” option uses a coefficient of drag to solve for the $x(t)$ and $y(t)$ equations discussed in section 2.1.2.3.

3.2.8 Output to Figure

It is possible the user will want to modify the plot of the projectile path, or view the velocity data. By placing a check next to the Output to Figure command, as in Figure 3.46, when the model is analyzed, the output will be draw in a separate figure window instead of inside the application. Now, the user can modify the plot from the Command Window. By selecting the Output Data command, as in Figure 3.47, the application will

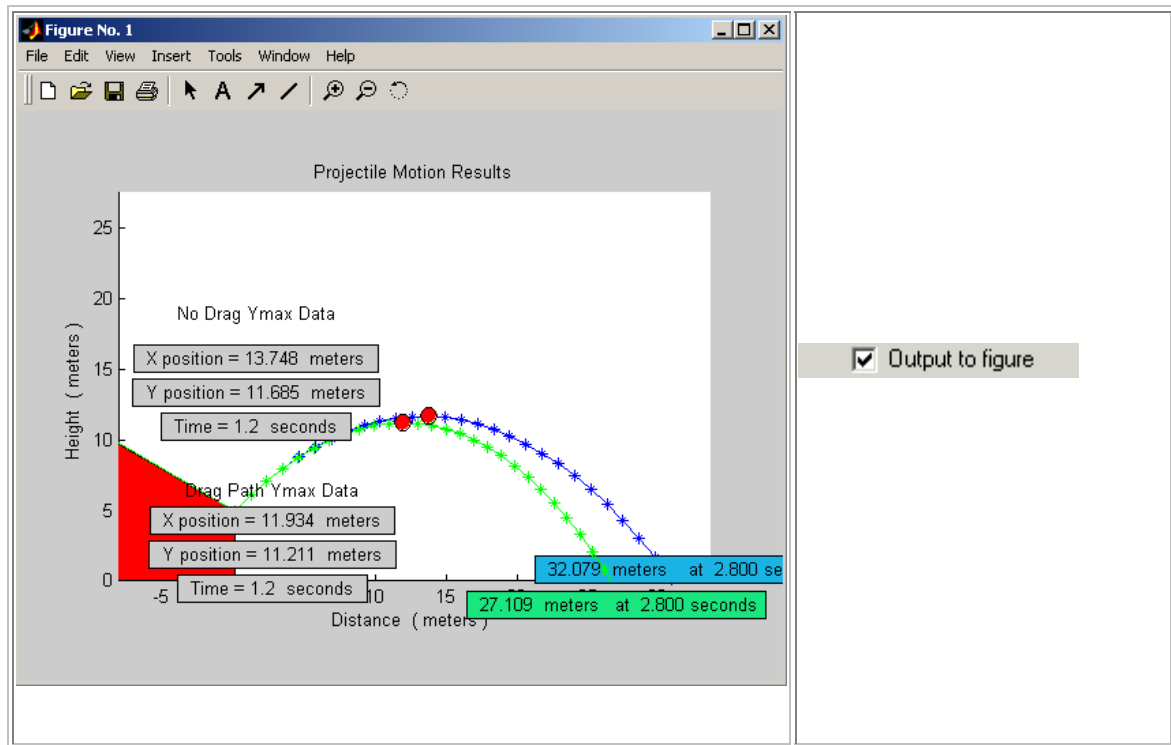


Figure 3.46 Ramp and Spring Launching Application, Figure Output

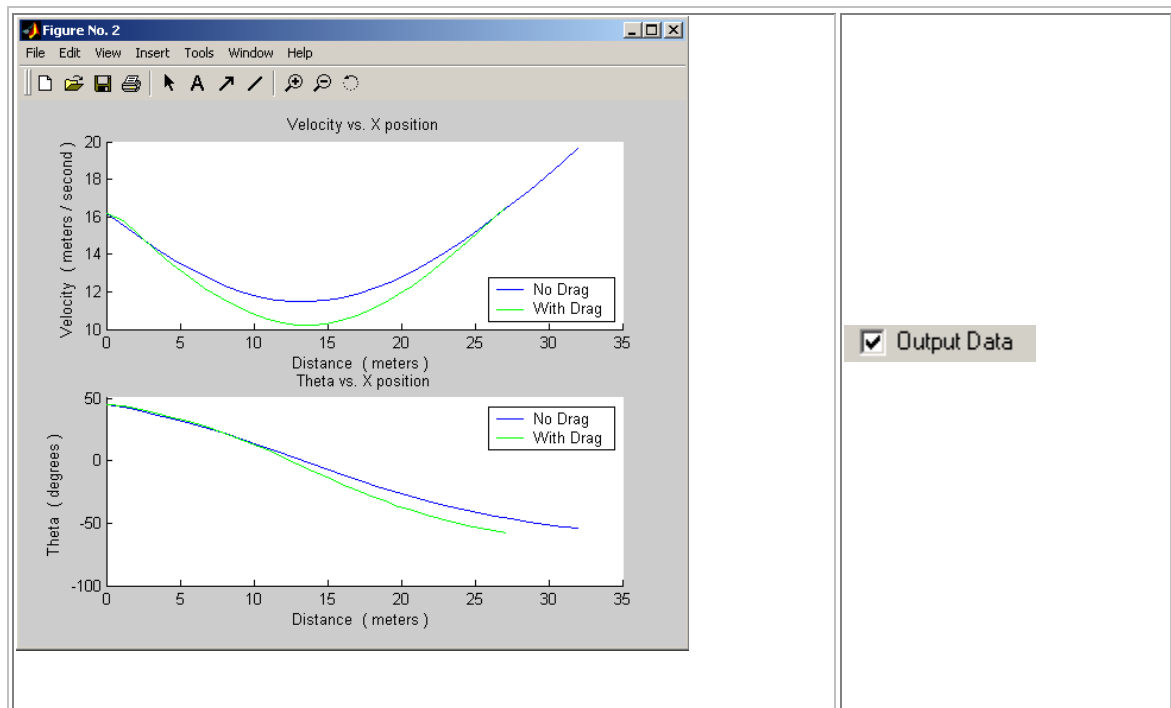


Figure 3.47 Ramp and Spring Launching Application, Data Output

make a plot of the velocity versus the x-position, a plot of the angle of the velocity versus the x-position, and write the raw data to an output file. The user can use the output file for additional analysis that is specialized to a particular team project.

3.2.9 Analysis Button

The analysis button, shown in Figure 3.48, initiates the application. Each variable coefficient is transferred to a variable inside the function. Based on the user selections, the application performs the necessary calculations. An example output is shown in Figure 3.49.



Figure 3.48 Ramp and Spring Launching Application, Analyze Button

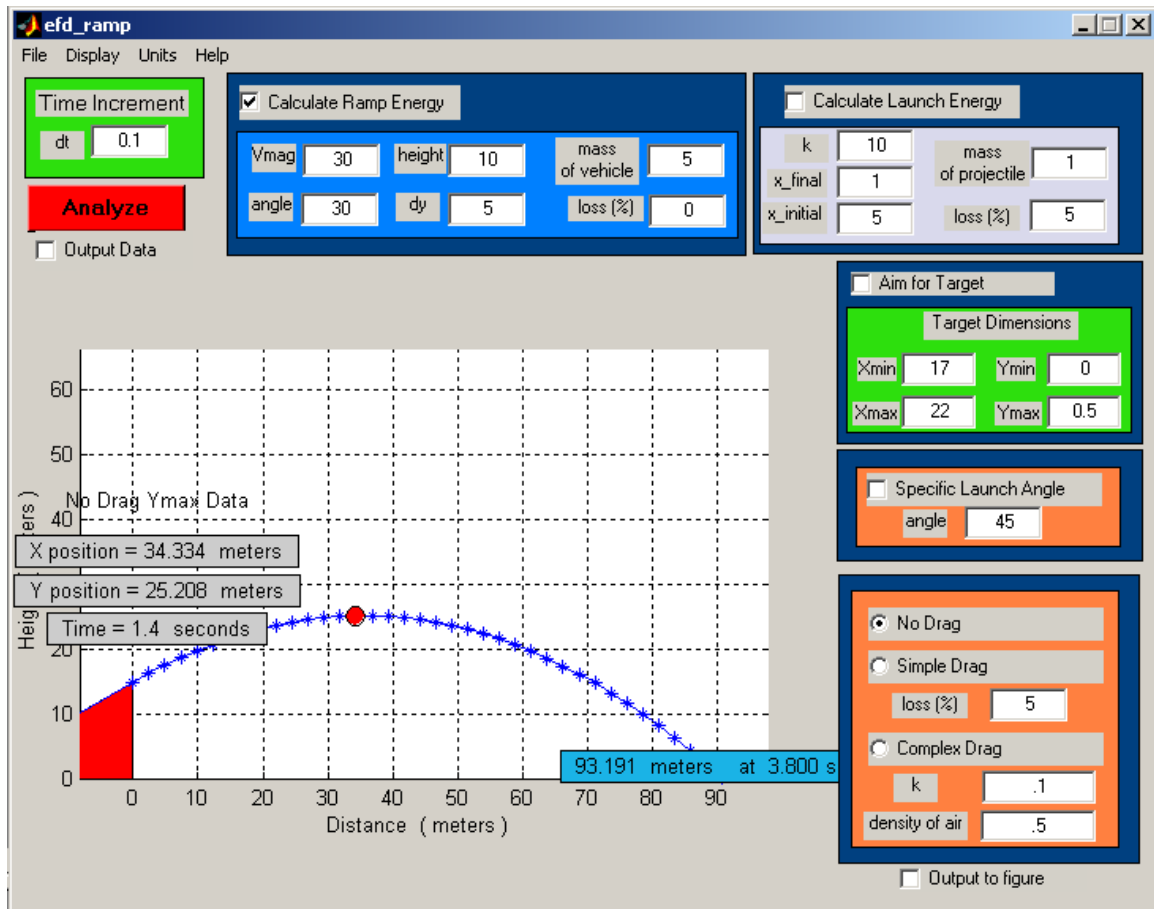


Figure 3.49 Ramp and Spring Launching Application, Analyze Output

3.2.10 Help File

The help file contains two main sections, feature information and walkthrough tutorials. The user can access the help file by clicking on the “Help” item on the toolbar, and then selecting “User Guide”. Once clicked, a web page help file appears. The information is similar to the above sections 3.2.1 through 3.2.9. If the users have a question about a particular menu, then they can open the help file and click on the topic of choice. The sample problems contain walkthrough tutorials, which help the users build their first model. The walkthroughs will be discussed in detail in chapter 4 as verification problems.

The help file’s homepage is shown in Figure 3.50. The user can find feature menu information by clicking on the list on the left, or by clicking on the picture of a particular feature menu.

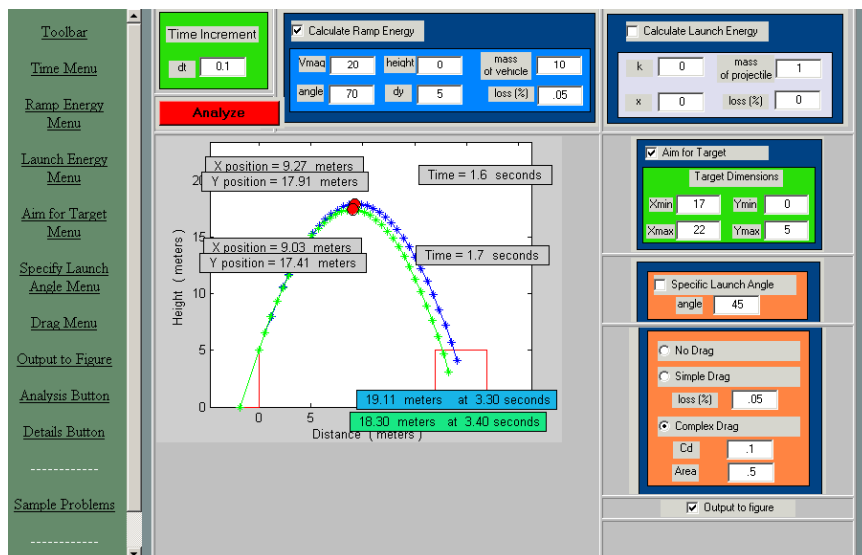


Figure 3.50 Ramp and Spring Launching Application, Help File

3.3 Swinging Projectile Application

The Swinging Projectile Motion Application, shown in Figure 3.51, solves a swinging energy transfer projectile motion and simple velocity projection problems. Similar to the Ramp Launched Projectile Application, an exit velocity is calculated from the energy transfer from the swing, the analysis is built from information from menus on the main draw area. The application is divided into eight main areas.

1. Toolbar Menu
 - a. File
 - b. Display
 - c. Units
 - d. Help
2. Properties Menu
3. Swing Energy Menu
4. Aim for Target Menu
5. Drag Menu
 - a. No Drag
 - b. Simple Drag
 - c. Complex Drag (calculation from drag coefficients)
6. Outputting Figures and Data
 - a. To the Main Draw Area
 - b. To a separate figure
 - c. Output data plots and results file

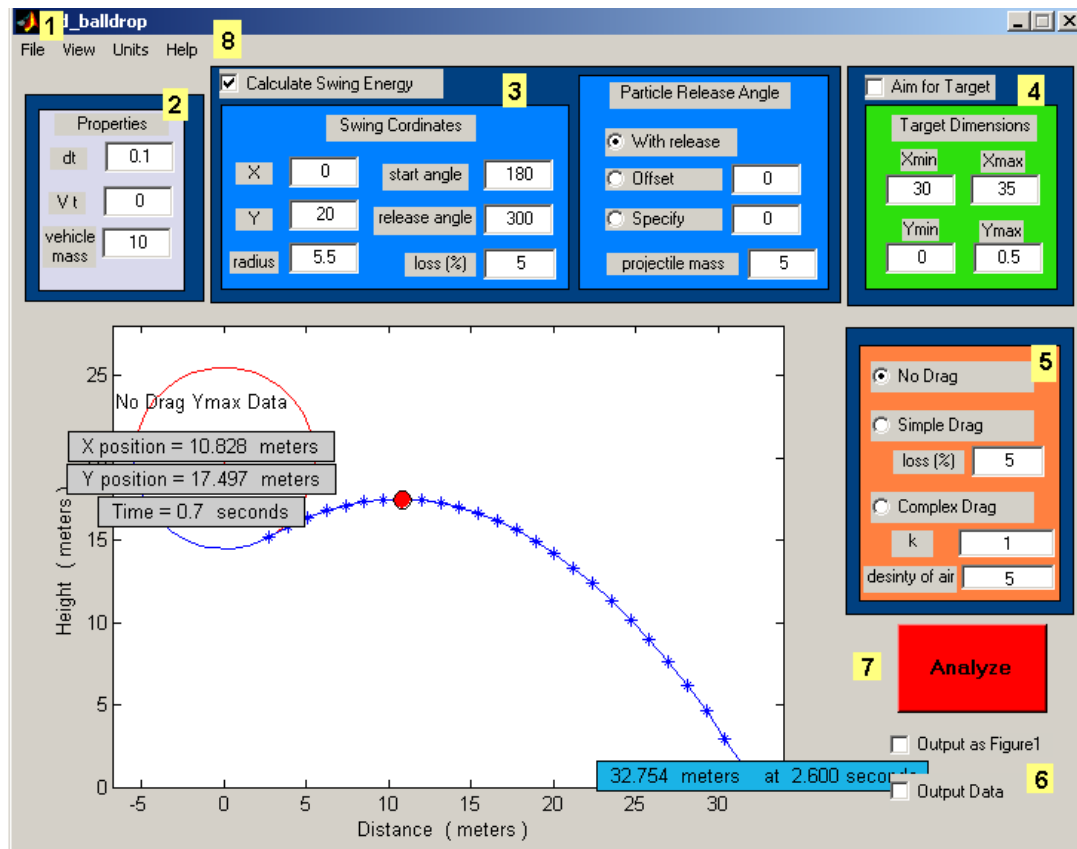
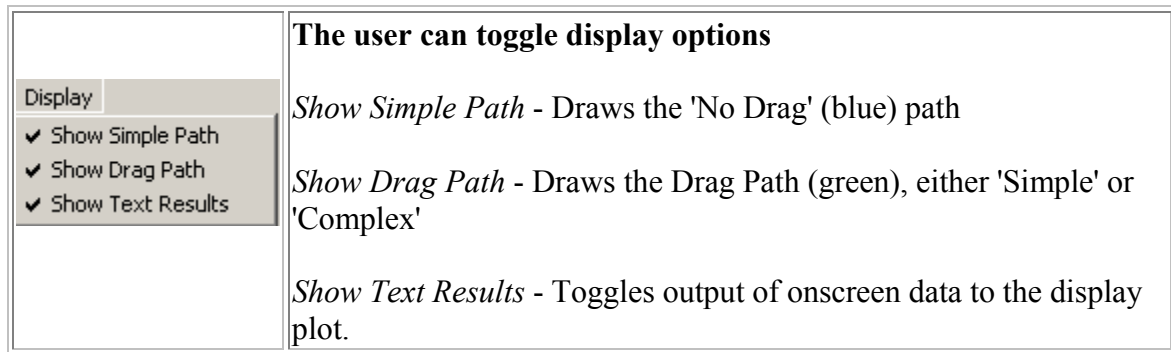


Figure 3.51 Swing Launching Application

7. Analysis Button
8. Help File

3.3.1 Toolbar

The Toolbar Menu, shown in Figure 3.52, contains all the file management and user option controls. The Toolbar consists of four headers, File, Display, Units, and Help.

Figure 3.52 Swing Launching Application, Toolbar**Figure 3.53 Swing Launching Application, Display Menu**

The file menu is only to close the application, or print the figure. There is not a need to save models, because the user entry is limited to a few pieces of data.

The display header, shown in Figure 3.53, controls the output to the screen, shown in Figure 3.54. It's possible to toggle each path on and off as well as the onscreen results.

The Units header, shown in Figure 3.55, informs the application which units to use for the entries. When the user selects a unit system, a figure appears, shown in Figure 3.56. The figure lists the units for each user entry and remains until the user closes the figure window. Optimally, the units would be next to each user entry, but there isn't enough room for all the menus and the units.

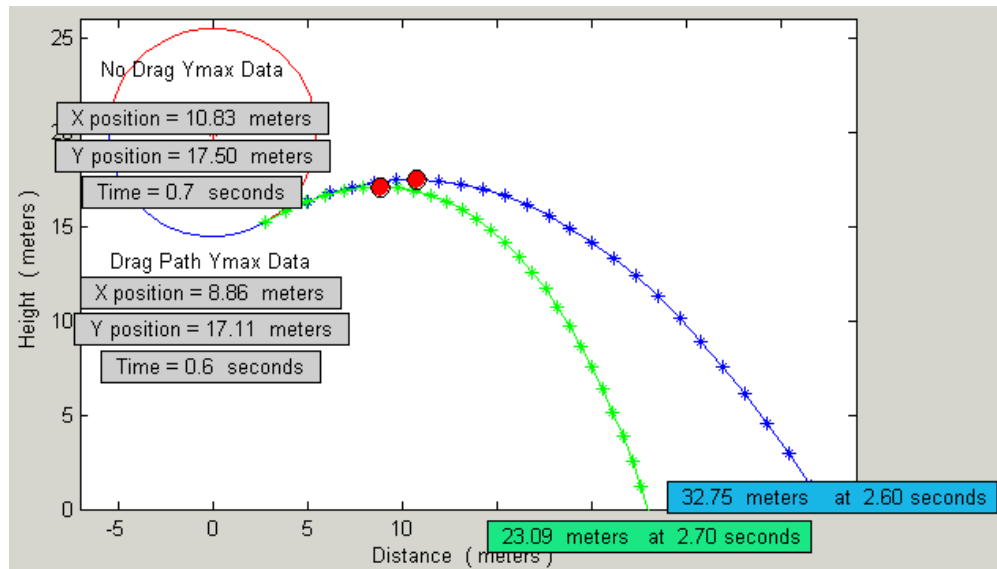


Figure 3.54 Swing Launching Application, Output

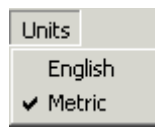


Figure 3.55 Swing Launching Application, Units Selection Menu

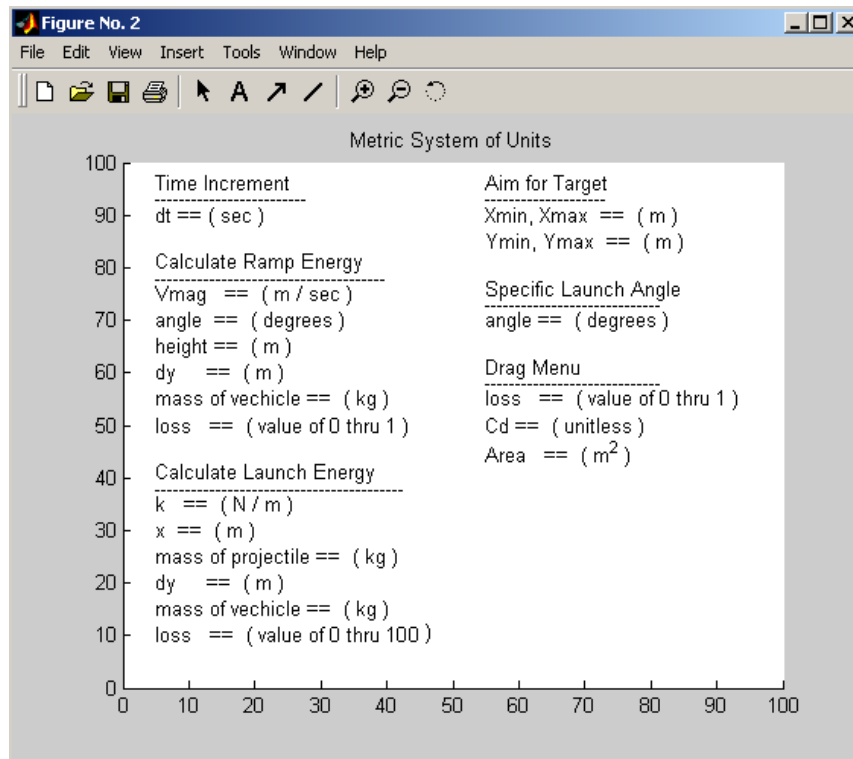


Figure 3.56 Swing Launching Application, Units Display

Properties dt 0.1 Vt 0 vehicle mass 10		dt - Enter the time increment.(seconds) Vt - Initial Velocity of vehicle at start angle $vehicle\ mass$ - total mass of swing vehicle and projectile

Figure 3.57 Swing Launching Application, Properties Menu

3.3.2 Properties Menu

In this menu, shown in Figure 3.57, the user can enter values for the time increment, an initial tangent velocity Vt (a value or zero), and the vehicle's mass. The vehicle mass is needed to calculate the change in potential energy between the start and release angle. The process of choosing a time increment is discussed in chapter 4.

3.3.3 Swing Energy Menu

This menu, shown in Figure 3.58, can simulate a swinging vehicle. Figure 3.59 corresponds to a projectile starting a 90° and swinging to 300° , and then releasing tangent to the release angle, the X and Y values specify the center of the swing arm. The radius of the swing arm is specified by the *radius* value. The *start angle* and *release angle* values control the initial and final position of the vehicle. The loss value will account for any losses attributed to the energy transfer. The release of the projectile can either be tangent

☒ Calculate Swing Energy													
Swing Coordinates <table> <tr> <td>X</td> <td>0</td> <td>start angle</td> <td>90</td> </tr> <tr> <td>Y</td> <td>30</td> <td>release angle</td> <td>300</td> </tr> <tr> <td>radius</td> <td>10</td> <td>loss (%)</td> <td>0</td> </tr> </table>		X	0	start angle	90	Y	30	release angle	300	radius	10	loss (%)	0
X	0	start angle	90										
Y	30	release angle	300										
radius	10	loss (%)	0										
Particle Release Angle <table> <tr> <td>☒ With release</td> <td></td> </tr> <tr> <td>☐ Offset</td> <td>0</td> </tr> <tr> <td>☐ Specify</td> <td>0</td> </tr> <tr> <td>projectile mass</td> <td>10</td> </tr> </table>		☒ With release		☐ Offset	0	☐ Specify	0	projectile mass	10				
☒ With release													
☐ Offset	0												
☐ Specify	0												
projectile mass	10												

Calculate Swing Energy <i>X, Y</i> - Coordinates of swing center point <i>radius</i> - Swing radius <i>start angle</i> - start angle <i>release angle</i> - release angle <i>loss</i> - amount of energy lost, 0 to 100.	Particle Release Angle <i>With release</i> - With release angle <i>Offset</i> - Offset from release angle <i>Specify</i> - A particular release <i>projectile mass</i> - Mass of projectile (for drag calculations)

Figure 3.58 Swing Launching Application, Swing Energy Menu

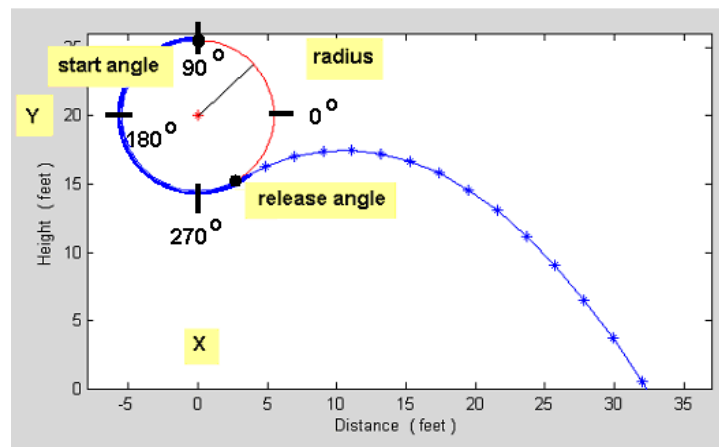


Figure 3.59 Swing Launching Application, User Input



Figure 3.60 Swing Launching Application, Aim for Target Menu

to the *release angle* value, or at an *offset* to the *release angle* value, or at a specified value.

3.3.4 Aim for Target Menu

The aim for target menu, shown in Figure 3.60, draws a rectangular target for the projectile. The user specifies a rectangular target using the values $Xmin$, $Xmax$, $Ymin$, and $Ymax$. When the projectile's path moves inside the target, the path terminates. If the path never moves inside the target, the path terminates when the project hits the ground at $y = 0$. Figure 3.61 shows the results with and without aiming for a target.

3.3.5 Drag Menu

The Drag Menu, shown in Figure 3.62, is for the user to select the type of loss associated with drag. There are three options, ignoring drag, a simple drag loss

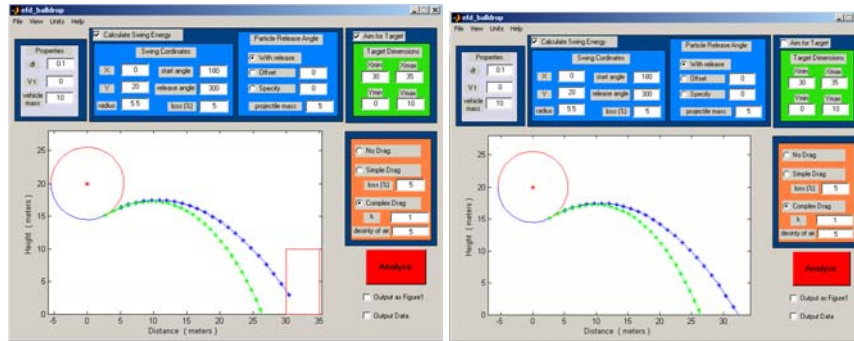


Figure 3.61 Swing Launching Application, With (left) and Without (right) Target Menu

	<i>No Drag</i> - Drag is not considered <i>Simple Drag</i> - Reduces the launch velocity by a percentage. <i>Complex Drag</i> - Calculates drag with respect to the coefficients and the instantaneous velocity.

Figure 3.62 Swing Launching Application, Drag Menu

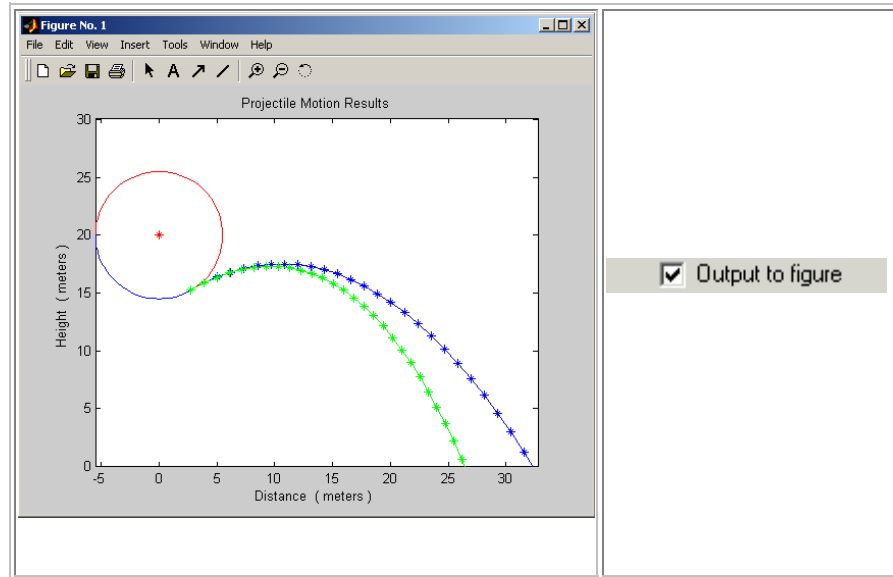


Figure 3.63 Swing Launching Application, Figure Output

calculation, and a coefficient drag calculation. The drag calculations are the same as outlined in section 3.2.7.

3.3.6 Output to Figure

It is possible the user will want to modify the plot of the projectile path, or view the velocity data. By placing a check next to the Output to Figure command, as in Figure 3.63, when the model is analyzed, the output will draw into a separate figure window instead of inside the application. Now, the user can modify the plot from the Command Window. By selecting the Output Data command, as in Figure 3.64, the application will make a plot of the velocity versus the x-position, a plot of the angle of the velocity versus

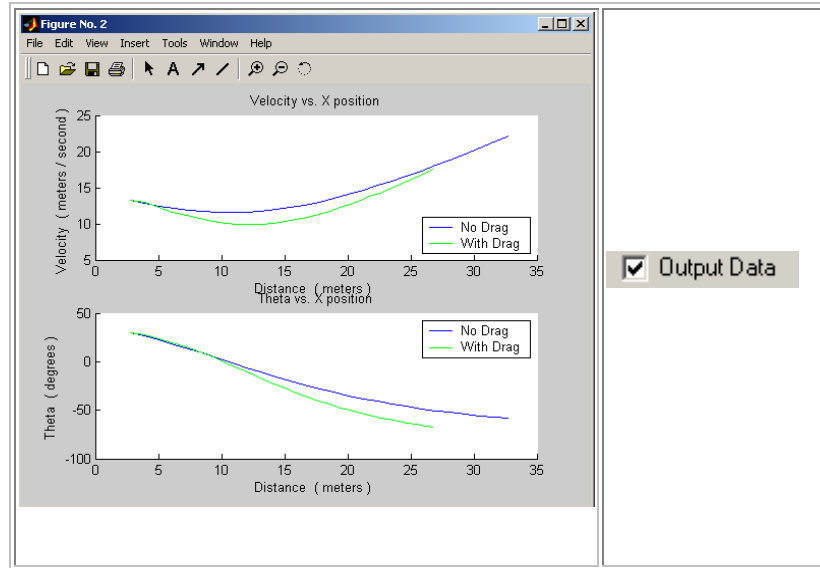


Figure 3.64 Swing Launching Application, Data Output

the x-position, and write the raw data to an output file. The user can use the output file for additional analysis that is specialized to a particular team project.

3.3.7 Analysis Button

The analysis button, shown in Figure 3.65, initiates the application. Each variable coefficient is transferred to a variable inside the function. Based on the user selections, the application performs the necessary calculations. An example output is shown in Figure 3.66.



Figure 3.65 Swing Launching Application, Analyze Button

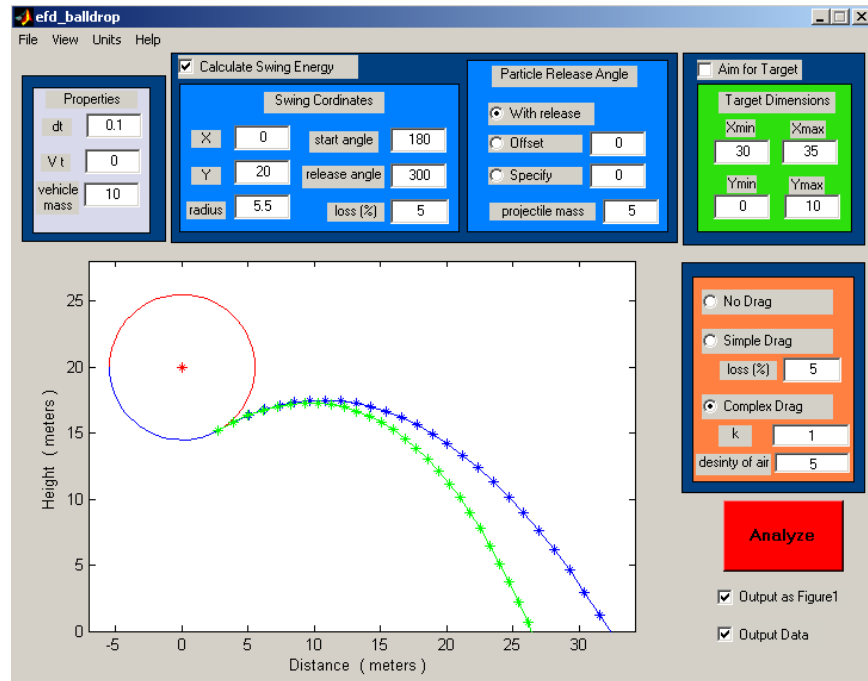


Figure 3.66 Swing Launching Application, Analysis Output

3.3.8 Help File

The help file contains two main sections, feature information and walkthrough tutorials. The user can access the help file by clicking on the “Help” item on the toolbar, and then selecting “User Guide”. Once clicked, a web page help file appears. The feature information is similar to the above listings. If the user has a question about a particular menu, then he can open the help file and click on the topic of choice. The sample problems contain walkthrough tutorials, which help the users build their first model. The walkthroughs will be discussed in detail in chapter 4 as verification problems.

The help file’s homepage is shown in Figure 3.67. The user can find feature menu information by clicking on the list on the left, or by clicking on the picture of a particular feature menu.

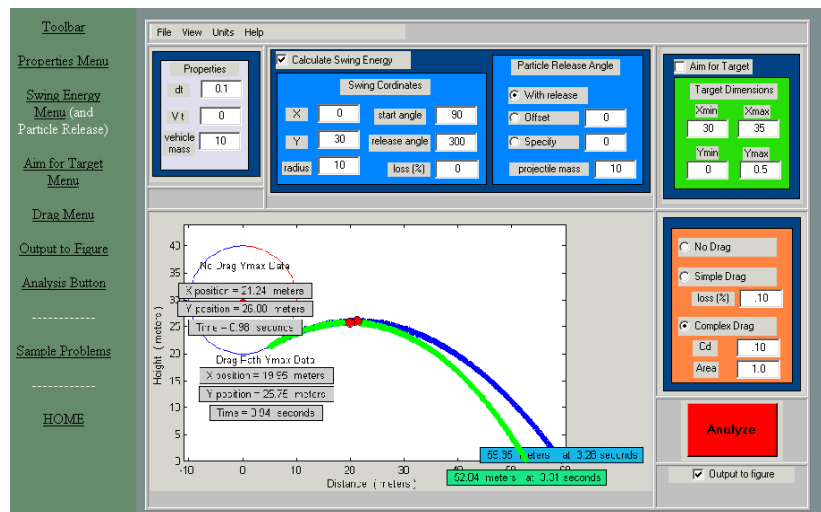


Figure 3.67 Swing Launching Application, Help File

Chapter 4 - Verification and Tutorials

Various example and homework solutions, from the Hibbeler Statics [15] and Dynamics [16] textbooks and the Boresi-Schmidt Engineering Mechanics [14] textbook, validate the calculations of each application and some examples are used as walkthrough tutorials. The solutions were chosen because they relate to a particular analysis for each application. The following is the list of verification problems used for each application.

- Truss Solver Application Verification Problems
 - Hibbeler Statics, Example 6-1, page 262 – Simple 2D Model
 - Hibbeler Statics, Example 6-2, page 263 – Simple 2D Model
 - Hibbeler Statics, Homework 6-30, page 279 – Complicated 2D Model
 - Hibbeler Statics, Homework 6-62, page 286 – 3D Model
- Ramp and Spring Energy Dynamics Verification Problems
 - Hibbeler Dynamics, Homework 14-28, page 179 – Ramp Energy Verification
 - Hibbeler Dynamics, Example 14-4, page 170 – Spring Energy Verification
 - Hibbeler Dynamics, Homework 12-86, page 45 – Projectile Motion Verification
 - Boresi / Schmidt Example 14-7, page 150 - Drag Verification from
- Swing Energy Dynamics Verification Problems
 - Hibbeler Dynamics, Homework 14-31, page 179 – Swing Energy Verification

4.1 Truss Solver Application Verification

The truss solver is capable of solving a wide range of truss types. Hibbeler Statics textbook examples 6-1 and 6-2 and homework problems 6-30 and 6-62 are used to verify the accuracy of the calculated answer and to serve as a walkthrough tutorial for using the application.

4.1.1 Hibbeler Example 6-1

Example 6-1 is a very simple truss to solve, (see Figure 4.1). Before the truss can be analyzed, it must be loaded into the application. This process includes specifying the joint numbers and coordinates, member connectivity, constraints, and forces. The member and reaction forces will be calculated from the entered information.

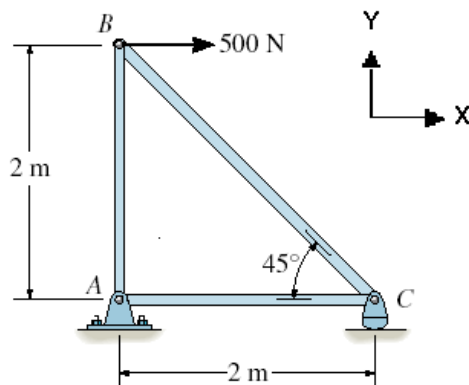


Figure 4.1 Hibbeler Example 6-1

4.1.1.1 Specify and Number Model Information

The first step for the user is to make a list of all joints, members, constraints, and forces, as shown in Tables 4.1, 4.2, 4.3, and 4.4, respectively. Even though this model is 2D, the information for every feature needs to be entered as a 3D model. For instance, this model is built on the XZ plane. All joints must have the same value for the y component, most likely zero. If the y values are not the same, the model does not exist on the same XZ plane. The constraint entry for the Y-axis degree of freedom must be free. If constrained, the model will be over constrained because no degree of freedom exists in the y direction on the XZ plane. The force value for FY must be zero, as well. The “Error Checking” “Validate 2D Plane” option informs the user if any of these errors exist in the 2D model, as discussed in section 3.1.1.

Table 4.1 Hibbeler Example 6-1 Joint Information

Joint Num	X-Pos (meters)	Y-Pos (meters)	Z-Pos (meters)
1	0	0	0
2	0	0	2
3	2	0	0

Table 4.2 Hibbeler Example 6-1 Member Information

Member Num	Joint 1	Joint 2
1	1	2
2	1	3
3	2	3

Table 4.3 Hibbeler Example 6-1 Constraint Information

Constraint Num	CX	CY	CZ	Joint Num
1	1	0	1	1
2	0	0	1	3

Table 4.4 Hibbeler Example 6-1 Force Information

Force Num	FX (Newtons)	FY (Newtons)	FZ (Newtons)	Joint Num
1	500	0	0	2

For constraints, a value of “1” represents a constraint being present and a value of “0” means the joint has a degree of freedom for that direction. For instance, if the constraint XYZ values are (1,0,1), then the constraints exist on the X and Z-axes, while no constraint is on the Y-axis. If the joint is completely unconstrained, then no information needs to be added. In fact, the application will report an error if the user attempts to add constraint information of (0, 0, 0) to a joint.

4.1.1.2 Enter Joint Information

Once the model information is organized, it can be entered into the application. First, ensure that the “Joint” feature is highlighted in the Feature Menu and “Add Feature” is selected in the Modify Menu, they do not have to be selected again. Enter the 3D coordinates for joint 1 (0,0,0) into the Enter Values Menu and then click Apply. Repeat the data entry for joints 2 (0,0,2) and 3 (2,0,0). Figures 4.2 and 4.3 visually progress through each joint for this truss.

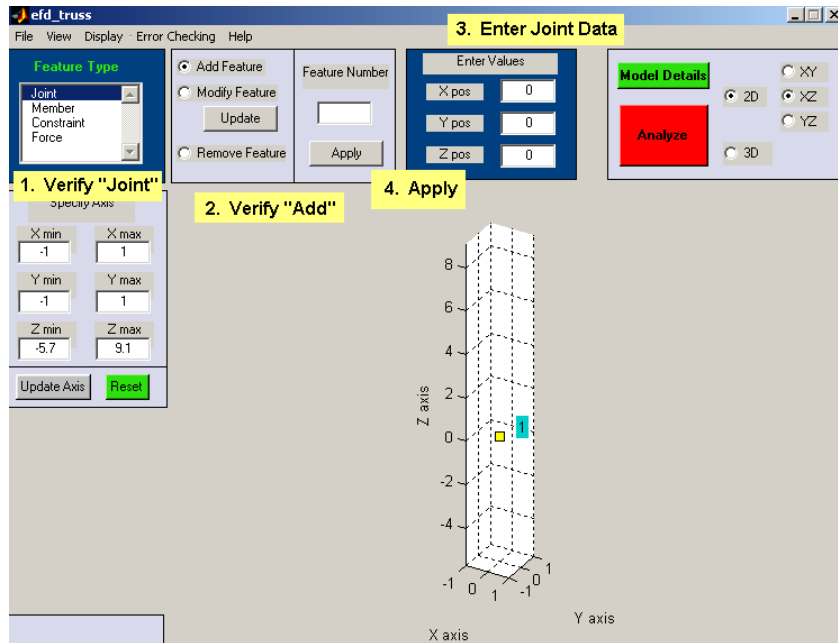


Figure 4.2 Adding joint 1 to the model

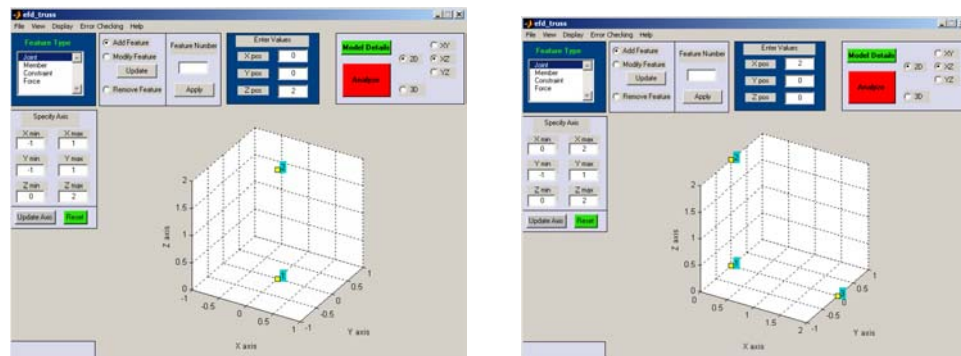


Figure 4.3 Results of adding joint 2 (left) and joint 3 (right) to the model

4.1.1.3 Enter Member Data

Next, select the “Member” feature in the Feature Menu and ensure “Add Feature” is selected in the Modify Menu. Enter the joint connectivity for member 1 (1,2) into the Enter Values Menu, and then click Apply. Repeat the data entry for members 2 (1,3) and 3 (2,3). The order of the connectivity does not have an affect on the application. For instance, member 1 can be (1,2) or (2,1). Figures 4.4 and 4.5 show the members being added to the model.

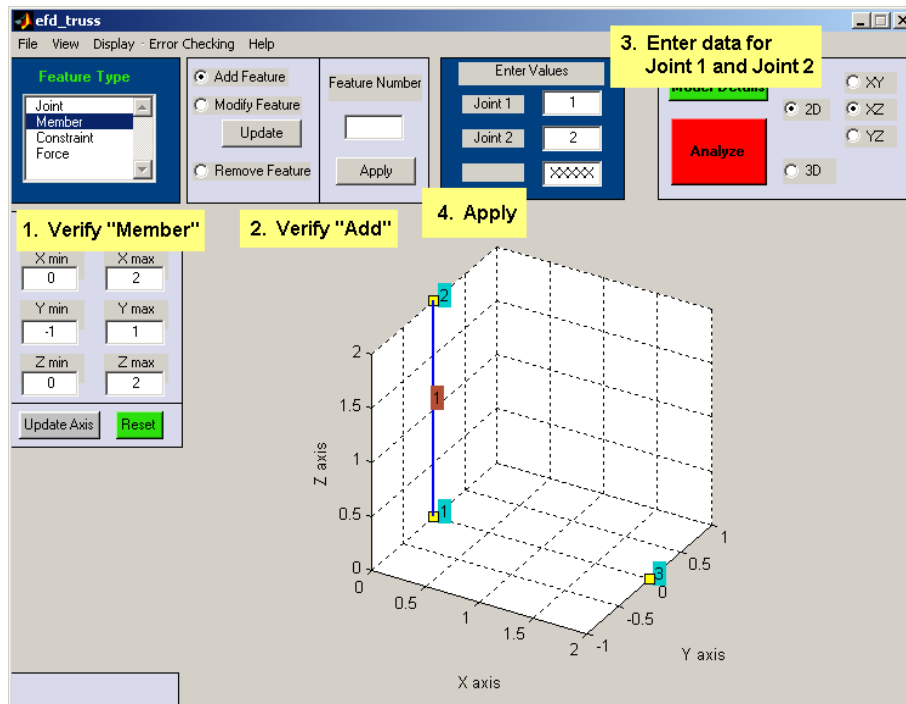


Figure 4.4 Steps to adding member 1 to the model

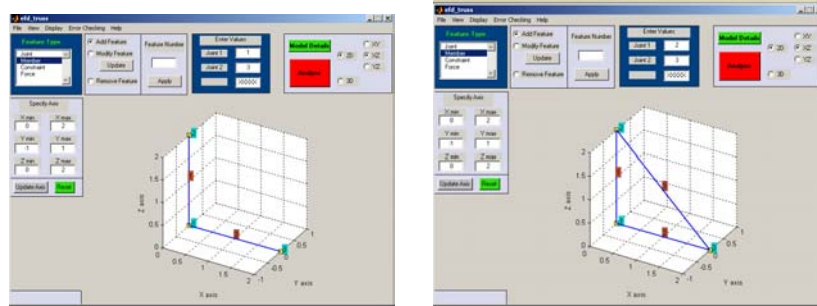


Figure 4.5 Results of adding member 2 (left) and member 3 (right) to the model

4.1.1.4 Enter Constraint Data

Select the “Constraint” feature in the Feature Menu and ensure “Add Feature” is selected in the Modify Menu. Enter the “Joint Number”, for the first constraint enter a “1” to attach the constraint data to joint 1. In the Modify Menu, enter the 3D constraint data for the constraint at joint 1 (1,0,1) into the Enter Values Menu, and then click Apply in the Modify Menu. The “Joint Number” in the Modify Menu attaches the constraint data to the specified joint. Repeat the data entry for the second constraint at joint 2 (0,0,1). Figures 4.6 and 4.7 show the constraints being added to the model. The constraint at joint 1 is a pin restraint, so it prohibits motion in two directions, the x and z-axes (1, 0, 1).

The constraint at joint 2 is a roller, and prohibits motion perpendicular to the roller, in this case, the z-axis (0, 0, 1). The total number of constraints is three; this value is the correct number of constraints for a 2D analysis.

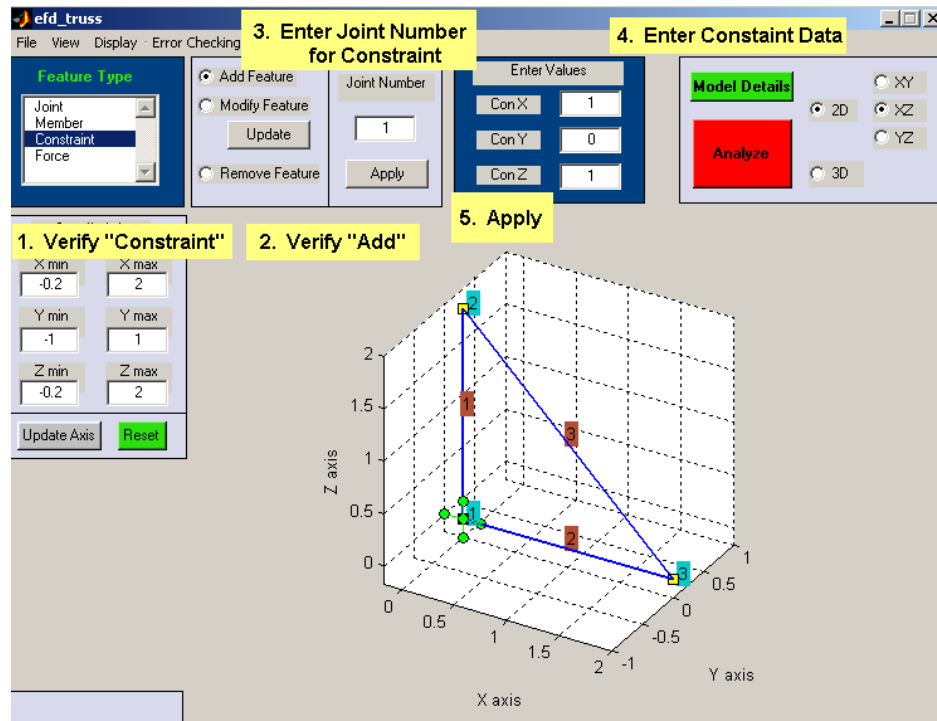


Figure 4.6 Steps to adding a new constraint

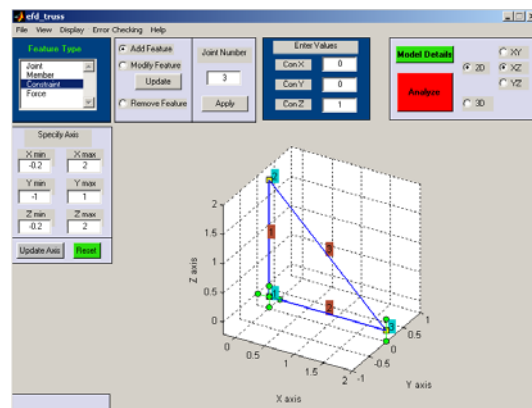


Figure 4.7 Results of adding all model constraints

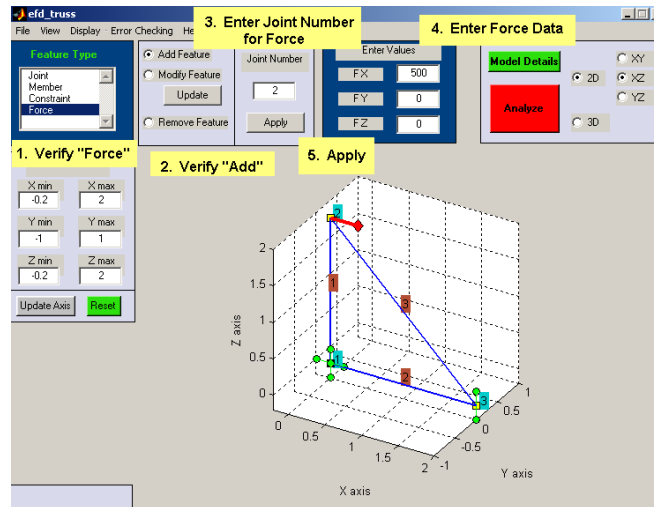


Figure 4.8 Steps to adding a new force

4.1.1.5 Enter Force Data

Select the Force option in the Feature Menu and “Add Feature” is selected in the Modify Menu. Enter the “Joint Number”, for this force enter a “2” to attach the force data to joint 2. In the Modify Menu, enter the 3D force data applied at joint 2 (500,0,0) into the Enter Values Menu, and then click Apply in the Modify Menu. Enter the 3D force data for the force at joint 2 (500,0,0) into the Enter Values Menu, and then click Apply. The “Joint Number” in the Modify Menu attaches the force data to the specified joint. Figure 4.8 shows the result of adding the force to the model.

4.1.1.6 Analyze

Once all the model information has been loaded, as in Figure 4.9, clicking the “Analyze” button will solve the model and display the results, as shown in Figure 4.10. It is important to make certain the 2D analysis plane matches the plane used for the model. If not, the analysis function will remove matrix entries that are required in order to solve the problem. If an incorrect matrix is used, the program will not calculate the correct answer. If the incorrect plane is set for this model, it usually results in “Inf” for the member and reaction force values.

4.1.1.7 View Results

By clicking the “Model Details” button, the model information is written to a text file and displayed in the Matlab Command Window, shown in Table 4.5. The application results match the answers from the Hibbeler textbook. This application accurately solves for member and reaction forces in simple 2D trusses.

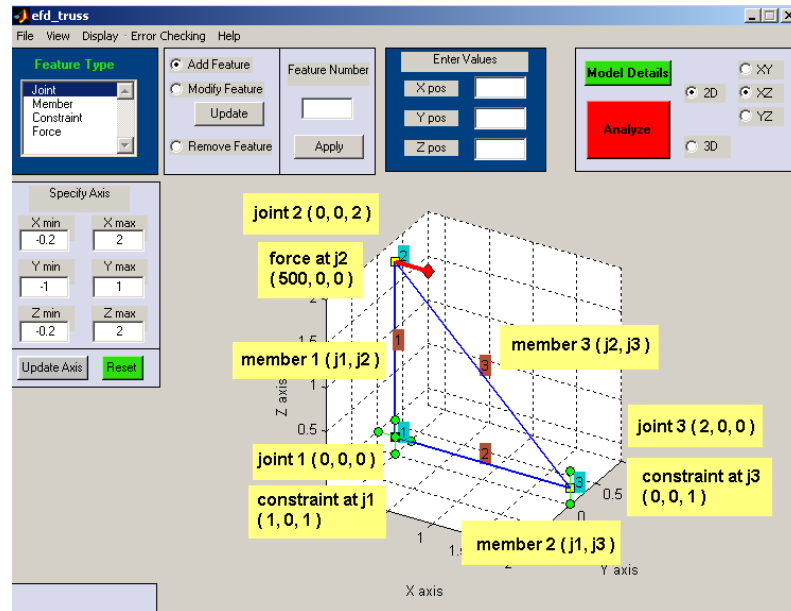


Figure 4.9 Example 6-1 complete model

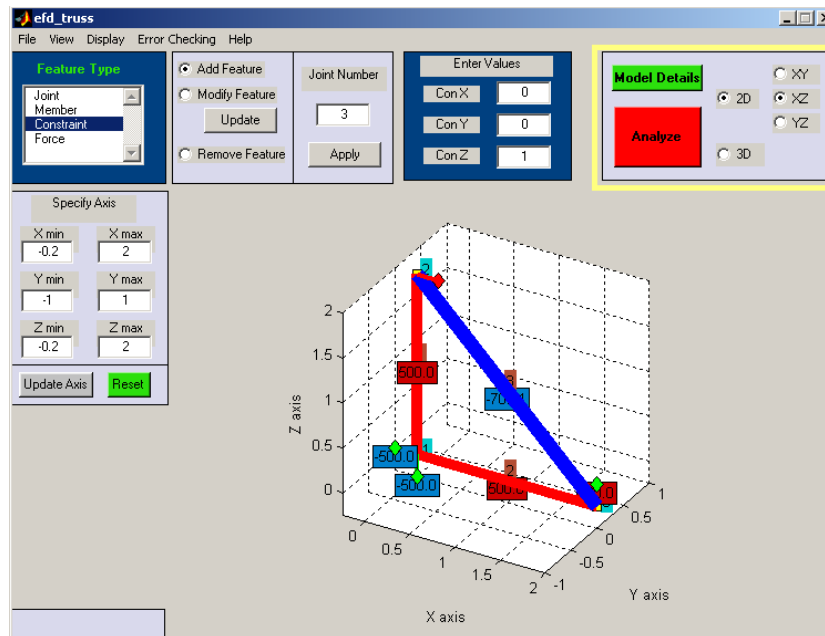


Figure 4.10 Example 6-1 analysis

Table 4.5 Hibbeler Example 6-1 Reaction Forces

Hibbeler Value (Newtons)	Force Value (Newtons)	Tension / Compression	Note	Member / Joint Num
500	500.00	T	Member	1
500	500.00	T	Member	2
707.1	707.11	C	Member	3
-500	-500.00		X constraint at Joint	1
-500	-500.00		Z constraint at Joint	1
500	500.00		Z constraint at Joint	3

4.1.2 Hibbeler Example 6-2

Similar to Example 6-1, the force in each member shown in Figure 4.11 is to be determined as well as the reaction forces from the constraints. The main purpose of this example is for verification of the program. While this example provides the users with an additional simple example to help familiarize them with the application, a detailed step-by-step walkthrough of this example is not provided.

The model information for all joints, members, constraints, and forces is shown in Tables 4.6, 4.7, 4.8, and 4.9, respectively and the completed model is shown in Figure 4.12. The analysis results are shown in Figure 4.13 and are listed in Table 4.10.

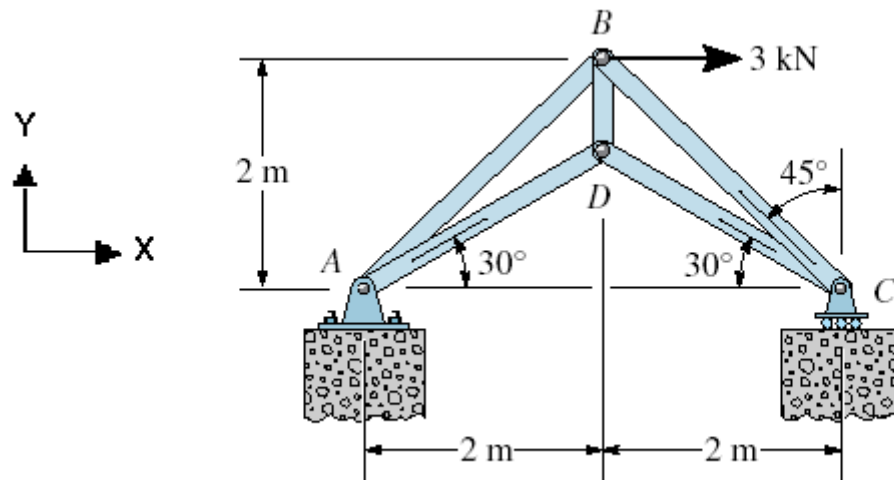


Figure 4.11 Hibbeler Example 6-2

Table 4.6 Hibbeler Example 6-2 Joint Information

Joint Num	X-Pos (meters)	Y-Pos (meters)	Z-Pos (meters)
1	0	0	0
2	2	0	2
3	4	0	0
4	2	0	1.15

Table 4.7 Hibbeler Example 6-2 Member Information

Member Num	Joint 1	Joint 2
1	1	2
2	3	2
3	3	4
4	2	4
5	1	4

Table 4.8 Hibbeler Example 6-2 Constraint Information

Constraint Num	CX	CY	CZ	Joint Num
1	1	0	1	1
2	0	0	1	3

Table 4.9 Hibbeler Example 6-2 Force Information

Force Num	FX (Newtons)	FY (Newtons)	FZ (Newtons)	Joint Num
1	3000	0	0	2

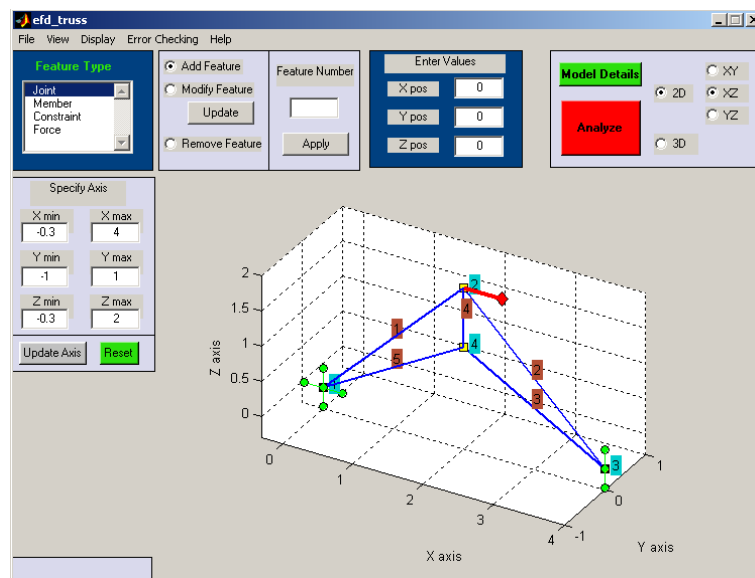


Figure 4.12 Hibbeler Example 6-2 entered into truss solver

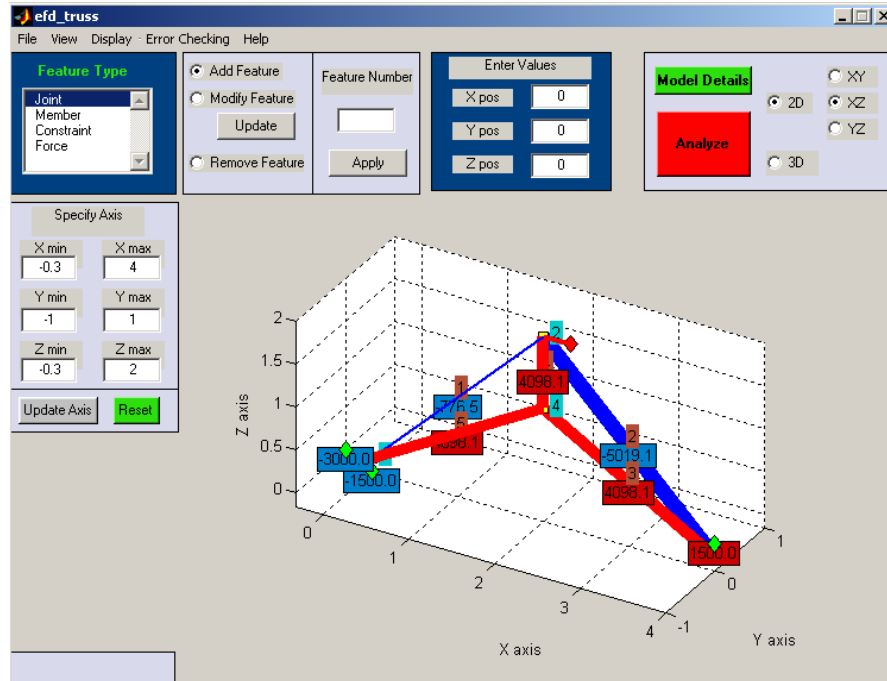


Figure 4.13 Hibbeler Example 6-2 Analysis Results

Table 4.10 Hibbeler Example 6-2 Analysis Results

Hibbeler Value (Newtons)	Force Value (Newtons)	Tension / Compression	Note	Member / Joint Num
776	776.45	C	Member	1
5020	5019.09	C	Member	2
4100	4098.07	T	Member	3
4100	4098.07	T	Member	4
4100	4098.07	T	Member	5
N/A	-3000.00		X constraint at Joint	1
N/A	-1500.00		Z constraint at Joint	1
N/A	1500.00		Z constraint at Joint	3

The application's calculations vary slightly from the provided solution. However, the Hibbeler text provided these answers to three significant figures. The application's results do match the Hibbeler solution to three significant figures. Once again, the application accurately solves for member and reaction forces in a simple 2D model.

4.1.3 Hibbeler Homework 6-30 / 6-31

Homework 6-30 and 6-31 represents a complicated 2D truss, (Figure 4.14). Homework 6-30 asks for the forces in members BC, HC, and HG, and homework 6-31 asks for forces in members GF, CF, and CD. The main purpose of this example is for verification of the program. Also, this example will give the users a complicated 2D example to help familiarize themselves with the application. It is not a detailed walkthrough.

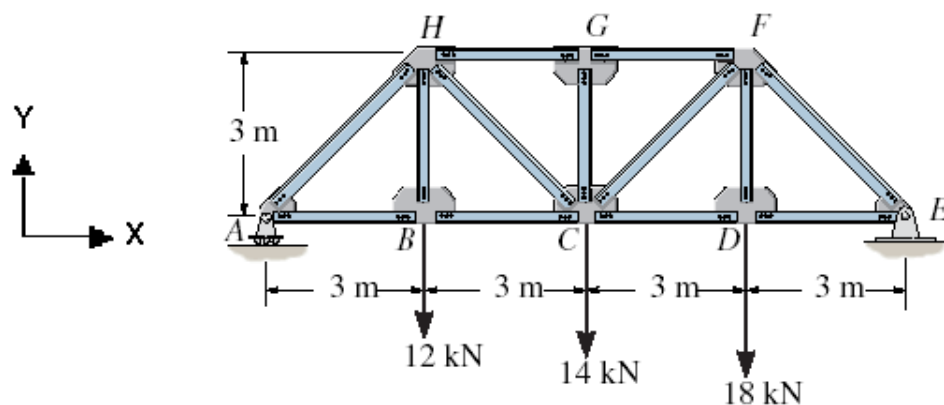


Figure 4.14 Figure for Hibbeler homework problems 6-30 and 6-31

The lists of model information for all joints, members, constraints, and forces are shown in Tables 4.11, 4.12, 4.13, and 4.14, respectively. The completed model is shown in Figure 4.15. The analysis results are shown in Figure 4.16 and Table 4.15. After successfully completing this practice problem, the user should be comfortable solving a moderately complicated 2D problem with this application.

The results are accurate to three significant figures compared to the provided solutions. Therefore, it can be concluded that this application correctly calculates the member and reaction forces of complicated 2D trusses within the precision used by the Hibbeler statics textbook.

Table 4.11 Hibbeler Homework 6-30 / 6-31 Joint Information

Joint Num	X-Pos (meters)	Y-Pos (meters)	Z-Pos (meters)
1	0	0	0
2	3	0	0
3	6	0	0
4	9	0	0
5	12	0	0
6	9	0	3
7	6	0	3
8	3	0	3

Table 4.12 Hibbeler Homework 6-30 / 6-31 Member Information

Member Num	Joint 1	Joint 2
1	1	2
2	3	2
3	3	4
4	5	4
5	5	6
6	4	6
7	7	6
8	6	3
9	7	3
10	7	8
11	3	8
12	2	8
13	1	8

Table 4.13 Hibbeler Homework 6-30 / 6-31 Constraint Information

Constraint Num	CX	CY	CZ	Joint Num
1	0	0	1	1
2	1	0	1	5

Table 4.14 Hibbeler Homework 6-30 / 6-31 Force Information

Force Num	FX (Newtons)	FY (Newtons)	FZ (Newtons)	Joint Num
1	0	0	-12000	2
2	0	0	-14000	3
3	0	0	-18000	4

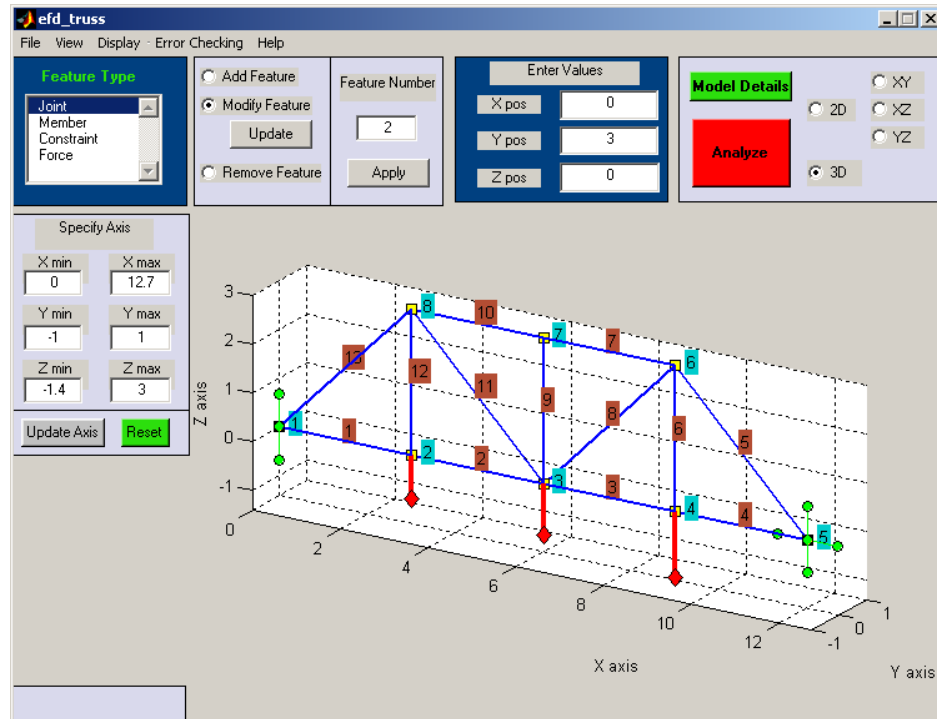


Figure 4.15 Hibbeler Homework 6-30 / 6-31 entered into truss solver

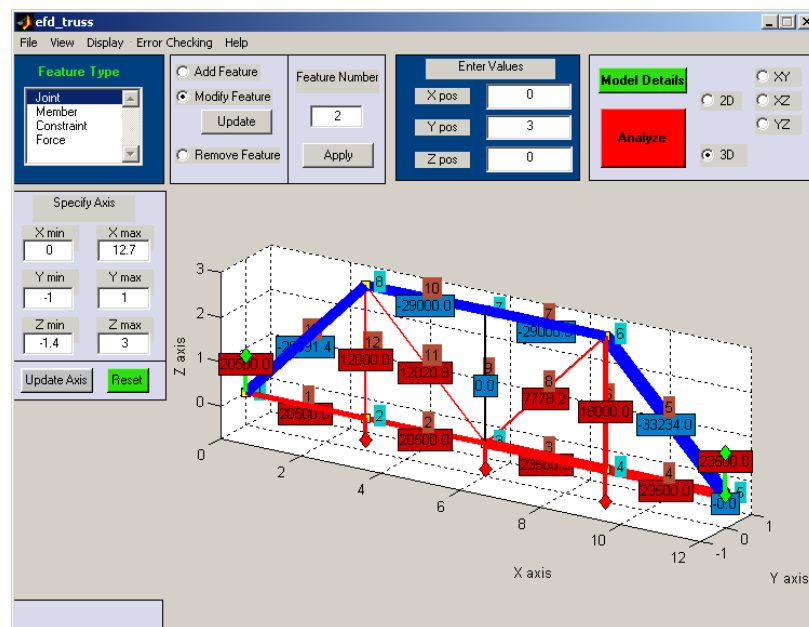


Figure 4.16 Hibbeler Homework 6-30 / 6-31 analysis results

Table 4.15 Hibbeler Homework 6-30 / 6-31 Results

Hibbeler Value (Newtons)	Force Value (Newtons)	Tension / Compression	Note	Member / Joint Num
N/A	20500.00	T	Member	1
20500	20500.00	T	Member	2
23500	23500.00	T	Member	3
N/A	23500.00	T	Member	4
N/A	33234.02	C	Member	5
N/A	18000.00	T	Member	6
29000	29000.00	C	Member	7
7780	7778.17	T	Member	8
N/A	0.00	-	Member	9
29000	29000.00	C	Member	10
12000	12020.82	T	Member	11
N/A	12000.00	T	Member	12
N/A	28991.38	C	Member	13
20500	20500.00		Z constraint at Joint	1
0	0.00		X constraint at Joint	5
23500	23500.00		Z constraint at Joint	5

4.1.4 Hibbeler Homework 6-62 / 6-63

These homework problems illustrate and validate the truss solver application's ability to solve 3D truss models. Homework 6-62 asks for the forces in members BE, DF, and BC, and homework 6-63 asks for the forces in members AB, CD, and ED, (Figure 4.17). The lists of model information for all joints, members, constraints, and forces are shown in Tables 4.16, 4.17, 4.18, and 4.19, respectively. The completed model is shown entered into the application in Figure 4.18.

Previously in the 2D trusses, the pin restricted motion in two directions, such as $(1, 0, 1)$. However, in 3D truss, the constraints must restrain the truss from translation and rotation along the three axes. A pin can constrain motion in all three directions, as in the constraint on joint 1 $(1, 1, 1)$. A pin constraint represents a fixed joint and constraints motion for all available degrees of freedom for that joint. In 2D models, joints have two degrees of freedom. For 3D model, joints have three degrees of freedom.

The analysis results are shown in Figure 4.19 and Table 4.20.

For this example, the results are accurate to four significant figures, except for member 1, this answer was provided to three significant figures and was correct to that precision. This application correctly calculates 3D trusses within the precision used by the Hibbeler statics textbook.

The truss solver is capable of solving complicated 3D trusses. After completing this example, the users should be able to use this application to analyze their 3D truss bridge designs for the *engage* team projects.

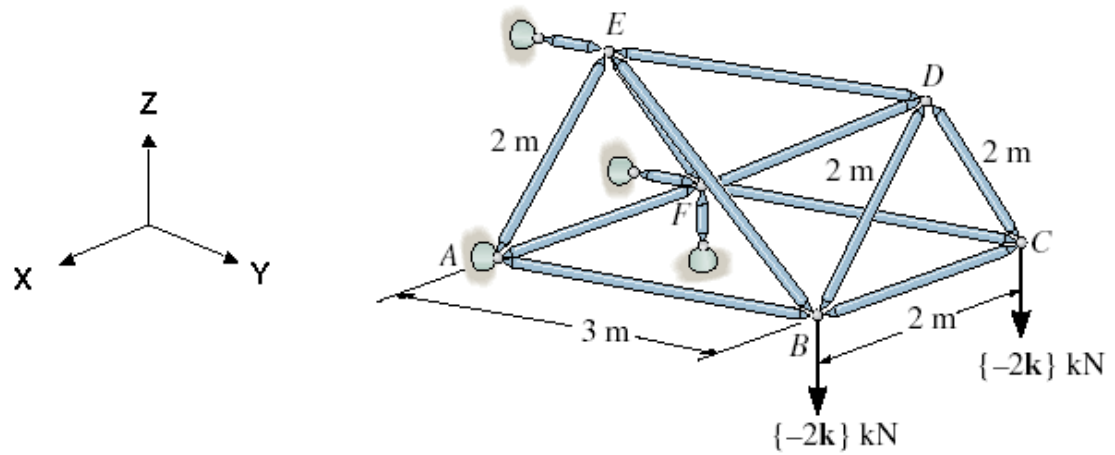


Figure 4.17 Figure for Hibbeler homework problems 6-62 and 6-63

Table 4.16 Hibbeler Homework 6-62 / 6-63 Joint Information

Joint Num	X-Pos (meters)	Y-Pos (meters)	Z-Pos (meters)
1	0	0	0
2	0	-3	0
3	2	-3	0
4	1	-3	1.73
5	1	0	1.73
6	2	0	0

Table 4.17 Hibbeler Homework 6-62 / 6-63 Member Information

Member Num	Joint 1	Joint 2
1	1	2
2	3	2
3	3	4
4	2	4
5	5	4
6	5	1
7	5	6
8	3	6
9	5	2
10	4	6
11	1	6

Table 4.18 Hibbeler Homework 6-62 / 6-63 Constraint Information

Constraint Num	CX	CY	CZ	Joint Num
1	1	1	1	1
2	0	1	0	5
3	0	1	1	6

Table 4.19 Hibbeler Homework 6-62 / 6-63 Force Information

Force Num	FX (Newtons)	FY (Newtons)	FZ (Newtons)	Joint Num
1	0	0	-2000	2
2	0	0	-2000	3

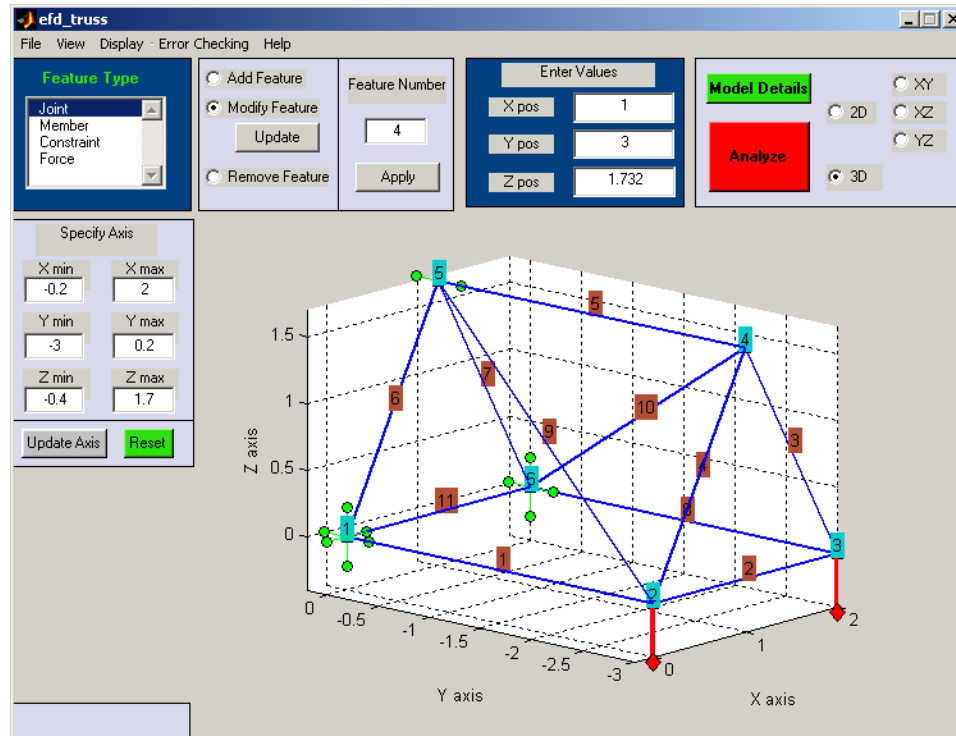


Figure 4.18 Hibbeler Homework 6-62 / 6-63 entered into truss solver

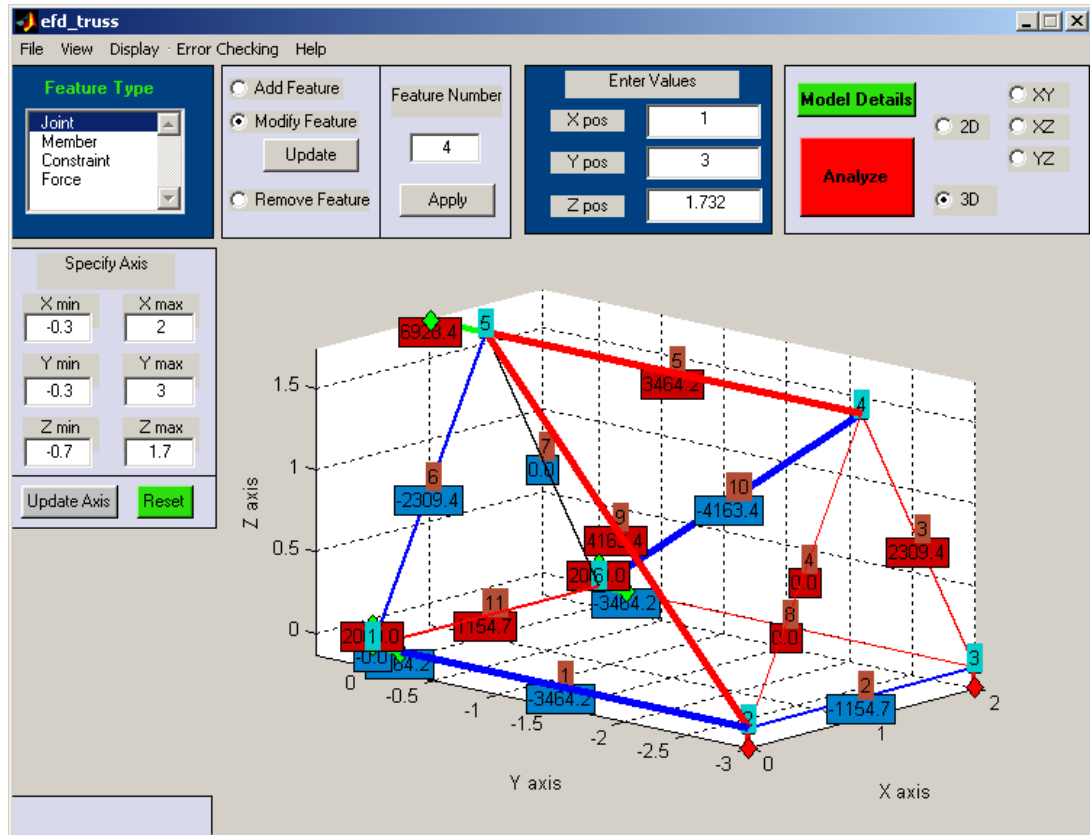


Figure 4.19 Hibbeler Homework 6-62 / 6-63 analysis results

Table 4.20 Hibbeler Homework 6-62 / 6-63 Results

Hibbeler Values (Newtons)	Force Value (Newtons)	Tension / Compression	Note	Member / Joint Num
3460	3464.20	C	Member	1
1150	1154.73	C	Member	2
2309	2309.42	T	Member	3
0	0.00	T	Member	4
N/A	3464.20	T	Member	5
N/A	2309.42	C	Member	6
N/A	0.00	-	Member	7
0	0.00	T	Member	8
4160	4163.43	T	Member	9
4160	4163.43	C	Member	10
N/A	1154.73	T	Member	11
N/A	0.00		X constraint at Joint	1
N/A	-3464.20		Y constraint at Joint	1
N/A	2000.00		Z constraint at Joint	1
N/A	6928.41		Y constraint at Joint	5
N/A	-3464.20		Y constraint at Joint	6
N/A	2000.00		Z constraint at Joint	6

4.2 Ramp Dynamics Verification Problems

For the Ramp Dynamics Application, various example and homework problems were used to validate the calculations of individual aspects of the application. Homework 14-28 demonstrates the application's ability to calculate the proper exit velocity as a function of the ramp height change. Example 14-4 validates the calculation of the conversion of stored linear spring energy to kinetic energy. Homework 12-86 demonstrates the effects of the time increment on the solution accuracy. The Boresi / Schmidt Example 14-7 verifies the simple and complex drag calculation.

4.2.1 Hibbeler Homework 14-28

The 2-lb brick slides down a smooth roof, at point A it has a velocity of 5 ft/s, (see Figure 4.20). Determine the speed of the block just before it leaves the surface at point B, the distance d from the wall to where it strikes the ground, and the speed at which it hits the ground. Table 4.21 contains the data that needs to be entered into the application.

The time increment of 0.00001 seconds was used because the Hibbeler solution for time also contains five significant figures. When choosing a time limit, the user must consider the situation of the analysis. The necessary time increment precision depends on the problem. A method can be applied to determine an acceptable time increment. First, analyze the model with a relatively large increment, like 0.1 seconds. Then, reduce the time increment by a factor of 10, until the calculation remains constant or the compute time becomes excessive relative to the importance of the problem. To illustrate this process, the results of time limits of 1, 0.1, 0.001, 0.0001, and 0.00001 seconds will be compared to the Hibbeler answer for five significant figures.

The remaining inputs are in the Ramp Energy Menu. The Vmag of 5 ft / s is the initial velocity of the sliding box at point A. The initial height is 45 feet and the box slides off the ramp when 30 feet from the ground. So, the dy value is -15 feet. Entering the angle requires quick calculation. This angle is 36.87 degrees CW from the x-axis, (Figure 4.21). The angle needs to be converted from the CW direction to the CCW direction. The ramp angle is 323.13 degrees CCW from the x-axis.

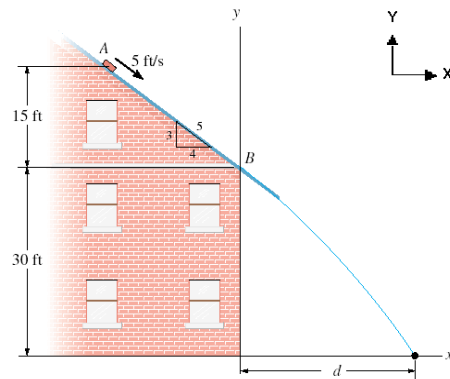


Figure 4.20 Figure for Hibbeler homework 14-28

Table 4.21 Ramp Energy Application Data

Variable	Value	Menu
dt	0.00001	Time Increment
Vmag	5	Ramp Energy
angle	323.13	Ramp Energy
height	45	Ramp Energy
dy	-15	Ramp Energy
Units	English	Toolbar

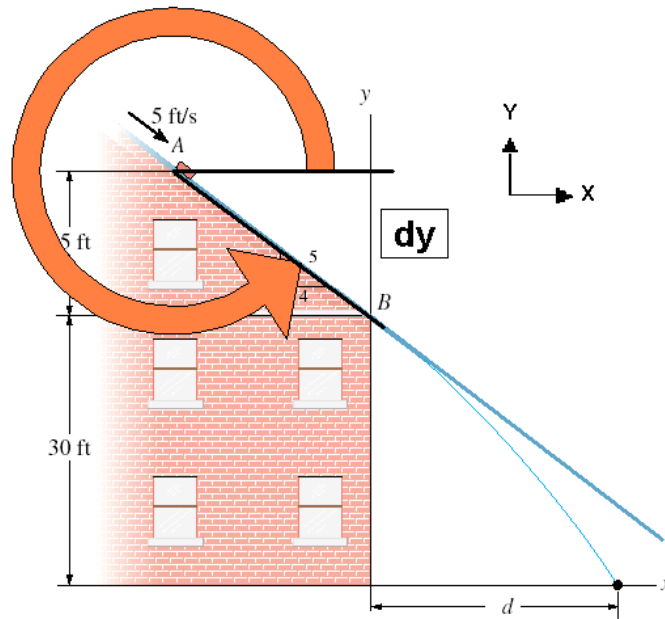


Figure 4.21 Ramp angle CCW from the x-axis

When the data is entered, the Time Increment Menu and Ramp Energy Menu should look like Figure 4.22. The Launch Energy Menu should be unchecked, because there is no spring energy in this problem. Once this data is entered, click “Analyze”. The results of the 0.00001 seconds model calculate the landing time accurately to five significant figures, as shown in Table 4.22 and Figure 4.23.

The results of the application for a time increment of 0.00001 seconds are shown in Table 4.23. The application calculates the correct answer to the precision of the Hibbeler textbook. However, the students input data for their team project will not be accurate to beyond two significant figures, so precision of the 0.01 seconds time increment is sufficient accuracy for the precision capabilities available for typical team projects.

Time Increment

dt 0.0001

☒ Calculate Ramp Energy

Vmag	5	height	45	mass of vehicle	2
angle	323.13	dy	-15	loss (%)	0

Figure 4.22 Hibbeler homework 14-28 data into Ramp Energy Menu

Table 4.22 Hibbeler homework 14-28 Time Increment Output

Time Increment (Seconds)	Time (Seconds)	% diff from Hibbeler answer	Compute time On P3 700 MHz
Hibbeler	0.89916	-	
1	1	10.08400%	>1 second
0.1	0.9	0.09333%	>1 second
0.01	0.90	0.09333%	>1 second
0.001	0.900	0.09333%	1 second
0.0001	0.8992	0.00445%	20 seconds
0.00001	0.89916	0.00000%	30 minutes

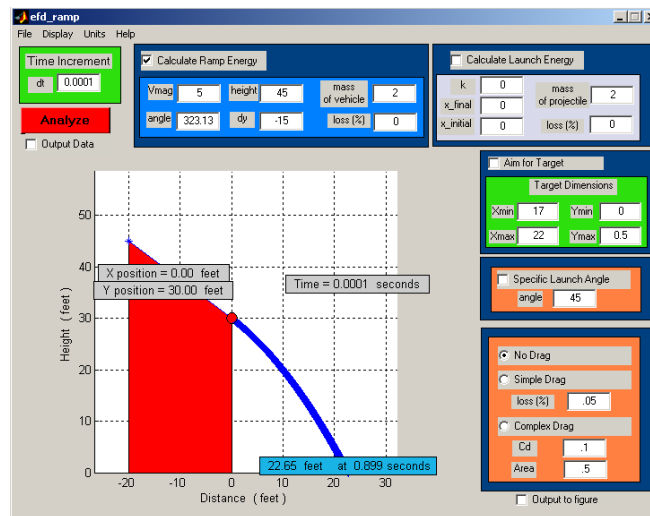


Figure 4.23 Application results for Hibbeler homework 14-28

Table 4.23 Ramp Energy Application Output

Variable	Application Answer	Text Answer
Velocity at B	$V_x = 25.184 \text{ ft / s}$ $V_y = -18.888 \text{ ft / s}$ $V_{\text{mag}} = 31.480 \text{ ft / s}$	$V_{\text{mag}} = 31.5 \text{ ft / s}$
Final Velocity	$V_x = 25.184 \text{ ft / s}$ $V_y = -47.841 \text{ ft / s}$ $V_{\text{mag}} = 54.064 \text{ ft / s}$	$V_{\text{mag}} = 54.1 \text{ ft / s}$
Length d	$d = 22.645 \text{ ft}$	$d = 22.6 \text{ ft}$
Time from B to ground	$t = 0.89916 \text{ sec}$	$t = 0.89916 \text{ sec}$

This application accurately calculates the launch velocity from a block with initial velocity sliding down a ramp. Also, the block's flight path is correct.

4.2.2 Hibbeler Example 14-4

The platform P shown in Figure 4.24 has a negligible mass and is tied down so that the 0.4-meter long cords keep a 1-meter long spring compressed 0.6 meter when nothing is on the platform. A 2-kg block is placed on the platform and released from rest after the platform is pushed down 0.1-meter. Determine the maximum height h the block rises in the air, measured from the ground. The data that needs to be entered in the Ramp Energy Menu and Launch Energy Menu is listed in Table 4.24.

In this case, the Ramp Energy Menu and Launch Energy must be checked to have the application account for both energy effects.

In the Ramp Energy Menu, the 2-kg block starts from rest so the V_{mag} value is 0 with the *height* value is 0.3 meters, and the dy value of 0.1 meters accounts for the

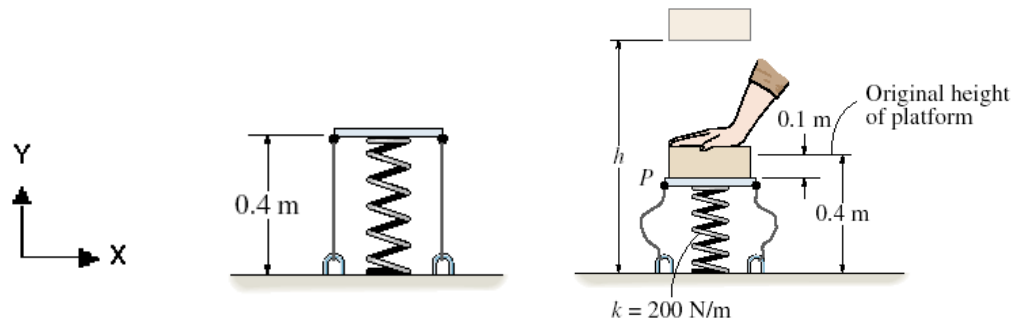


Figure 4.24 Hibbeler Example 14-4, Platform, unloaded (left) and loaded (right)

Table 4.24 Hibbeler Example 14-4 data entered into application

Variable	Value	Menu
dt	0.01	Time Increment
k	200	Spring Energy
x_final	0.7	Spring Energy
x_initial	0.6	Spring Energy
mass of projectile	2	Spring Energy
Loss	0	Spring Energy
Vmag	0	Ramp Energy
angle	90	Ramp Energy
height	0.3	Ramp Energy
dy	0.1	Ramp Energy
mass of vehicle	2	Ramp Energy
Loss	0	Ramp Energy
units	Metric	Toolbar

The screenshot shows a software interface for calculating energy and time. It is divided into three main sections:

- Time Increment:** A green box containing a label "Time Increment", a text input field "dt" with the value "0.01", a red "Analyze" button, and a checkbox "Output Data" which is currently unchecked.
- Calculate Ramp Energy:** A blue box with a checked checkbox "Calculate Ramp Energy". It contains several input fields: "vmag" (0), "height" (.3), "mass of vehicle" (2), "angle" (90), "dy" (.1), and "loss (%)" (0).
- Calculate Launch Energy:** A light blue box with a checked checkbox "Calculate Launch Energy". It contains input fields for "k" (200), "x_final" (.7), "x_initial" (.6), "mass of projectile" (2), and "loss (%)" (0).

Figure 4.25 Hibbeler Example 14-4 Time Increment, Ramp Energy, and Spring Energy Menu Settings

upward movement of the spring. The angle is set to 90. The mass value is 2-kg. As an assumption, the loss associated with the kinetic energy transfer is 0%.

In the Launch Energy Menu, the linear spring has a spring constant k of 200 N/m and is compressed by 0.7 meters and can expand to 0.6 meters. Therefore, the $x_{initial}$ value is 0.6 meters and the x_{final} value is 0.7 meters because the spring is initially compressed 0.6 meters and the initial applied energy to the spring resulted in a final compression of 0.7 meters before the block is released. The *mass of the projectile* value must be set to 2-kg. There is no assumed loss associated with the spring, so the loss value is 0%. To user inputs to solve the Hibbeler example are shown in Figure 4.25.

For this problem, a time increment of 0.01 calculates the height value to the precision from the Hibbeler textbook. An incremental time decrease starting at 0.1 seconds similar to homework problem 14-28 determines the necessary time increment. The results are shown in Figure 4.26 and Table 4.25. The calculation is accurate to the precision of the Hibbeler textbook. This application accurately calculates a linear spring launched projectile with a reduction in kinetic energy from a change in height.

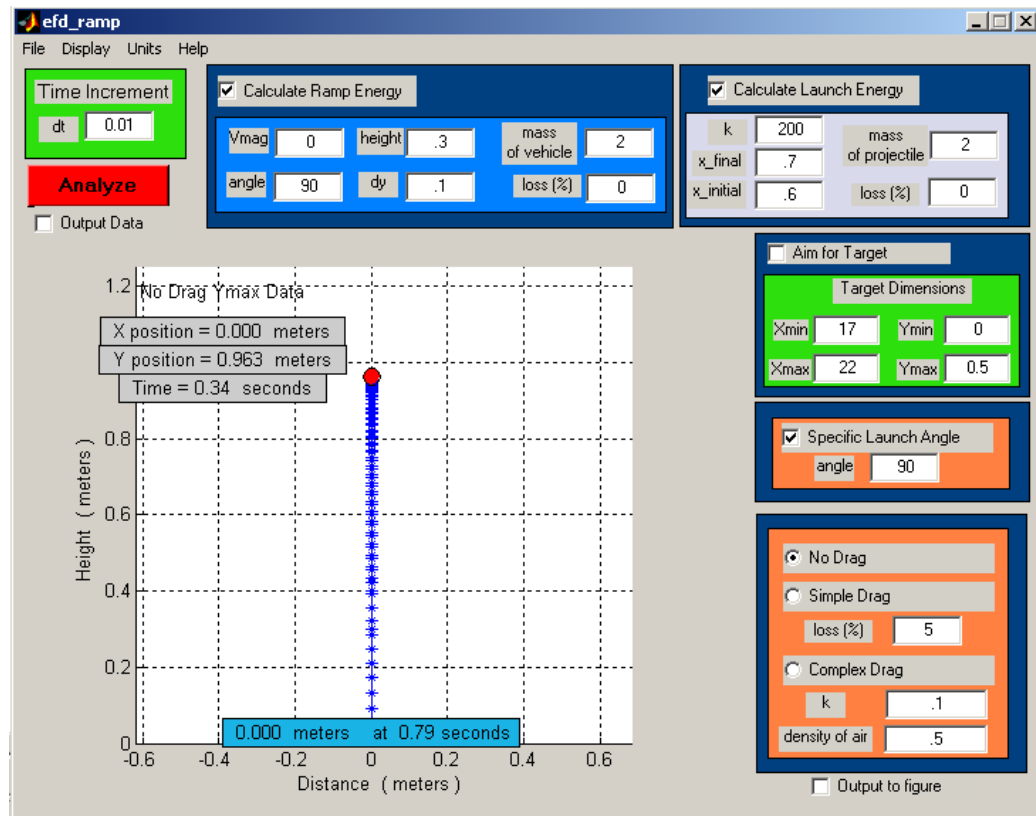


Figure 4.26 Hibbeler Example 14-4 application results

Table 4.25 Hibbeler Example 14-4 application results

Variable	Application Answer	Text Answer
Max height (meters)	0.963	0.963

4.2.3 Hibbeler Homework 12-86

The fireman standing on the ladder wishes to direct the flow of water from his hose to the fire at point B, (Figure 4.27). Determine the two possible angles θ_1 and θ_2 that send the water into the upper corner of the window. Water exits from the hose at $V_A = 300 \text{ ft / s}$. Table 4.26 contains the data that needs to be entered into the application.

A value of 0.001 seconds was used for the time increment. However, the time increments of 1, 0.1, 0.01, and 0.001 seconds were studied. The initial velocity V_{mag} was defined in the problem to be 300 feet / sec. The direct height is not specified in the problem. However, the water hose is thirty feet above the top of the window. The water does not use a ramp; therefore, the dy value is 0. To account for the window, the Aim for Target Menu needs to be checked and the upper left hand corner of the target must start and the coordinates (60, 10), shown as X_{min} and Y_{max} . The lower right hand corner is not specified by the problem. For this analysis, (65, 0) were used as the X_{max} and Y_{min} values. To best model this problem, the projectile's path must hit as close to the upper left hand corner as possible.

The application is not capable of calculating the launching angles from user data. Instead, the Hibbeler solutions will be used for the desired angles. If the student has not calculated the angles, he can try various launching angles until the path intersects with the upper corner of the target.

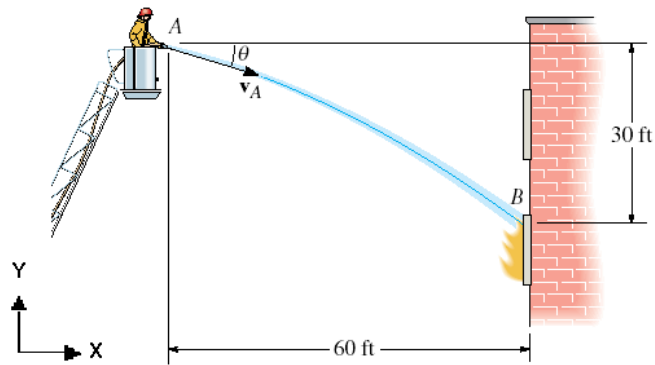


Figure 4.27 Hibbeler Homework 12-86 Figure

Table 4.26 Hibbeler Homework 12-86 data entered into application

Variable	Value	Menu
dt	0.001	Time Increment
Vmag	300	Ramp Energy
Angle (CCW from x-axis)	334 and 89.4 (89.388)	Ramp Energy
height	40	Ramp Energy
dy	0	Ramp Energy
units	English	Toolbar
Target	Xmin = 60 Xmin = 65 Ymin = 0 Ymax = 10	Aim for Target

This homework problem has two possible mathematical solutions of 26 degrees CW and 89.4 degrees CCW of the x-axis, shown in Figure 4.28. The direct path of 26 degrees (334 degrees CCW) is accurate compared to the Hibbeler solution, as in Figure 4.29. However, with the Hibbeler provided precision, the lofting angle of 89.4 CCW solution falls short of the target, as in Figure 4.30.

The projectile is supposed to land into the top, left corner of the target, as is the case, when the launch angle is 334 degrees. However, because of the high arcing path when the angle is 89.4 degrees, the projectile does not hit the target as predicted by the Hibbeler solution. Figure 4.31 is zoomed to a view scaled to fit the projectile's path near the target. The projectile should have hit the target, but instead it landed 1.049 feet short. When the precision of the launch angle is increased from 89.4 to 89.388, the application is capable of calculating a more accurate answer, shown in Figure 4.32. In fact, as the time increment is decreased, the lower precision angle decreases in accuracy, whereas the more precise angle calculation increases in accuracy, in Table 4.27.

The *engage* team projects do not require the precision necessary to calculate the exact high arc solution. However, the precision of the analysis is directly related to the precision of the initial angle. On the direct path of 334 degrees, the projectile hits the target in 0.23 seconds; the projectile misses the target for the time increments of 0.1 and 1 second, as shown in Figure 4.33. The path goes through the target in the upper left hand corner, but the time increment did not calculate for an entry while inside the target. The size of the target is small relative to the length traveled over a time increment. A smaller time increment fixes this issue.

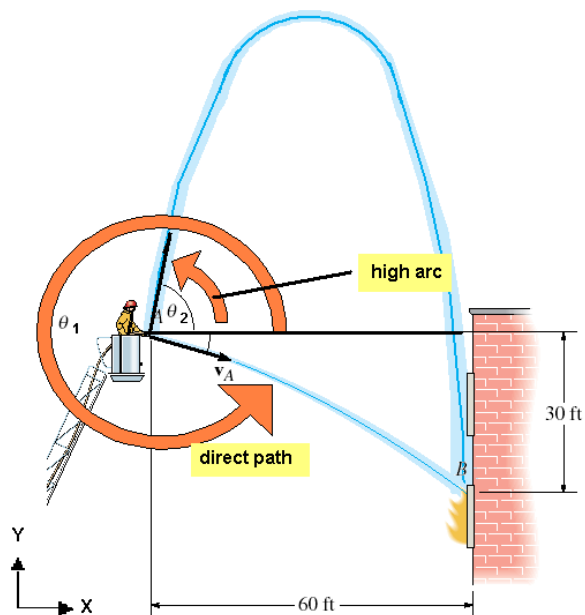


Figure 4.28 Possible launch angles CCW from x-axis

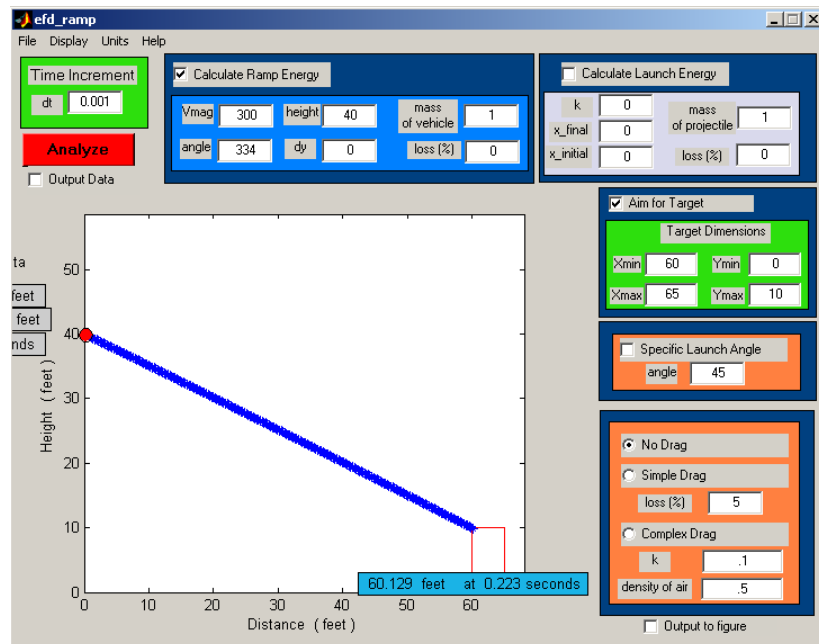


Figure 4.29 Hibbeler Homework 12-86 "Low Arc" solution

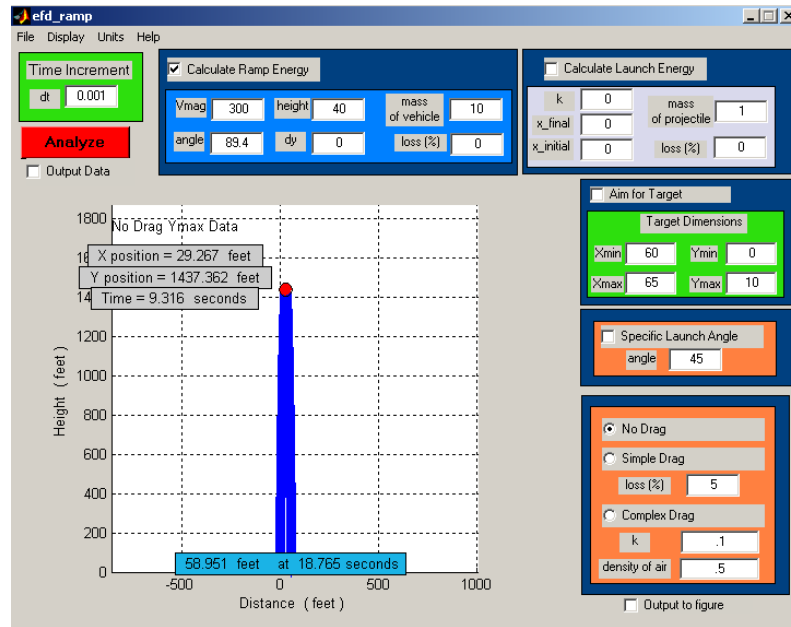


Figure 4.30 Hibbeler Homework 12-86 “High Arc” solution

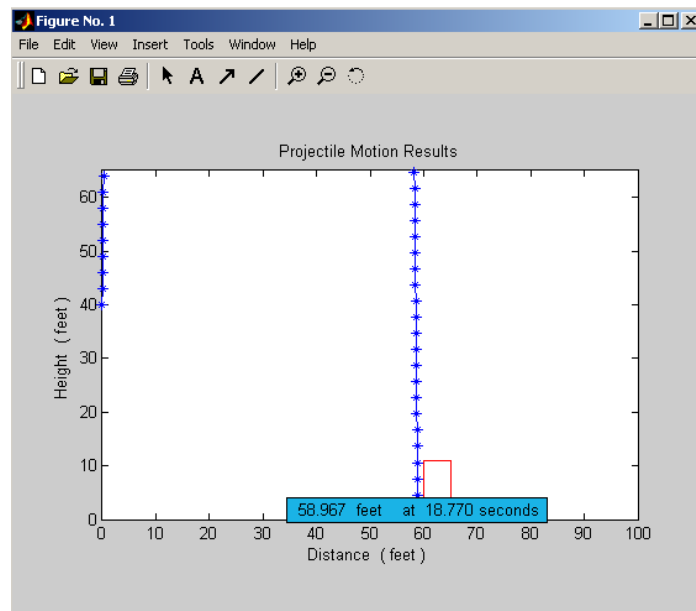


Figure 4.31 Hibbeler Homework 12-86 “High Arc” solution focused on target

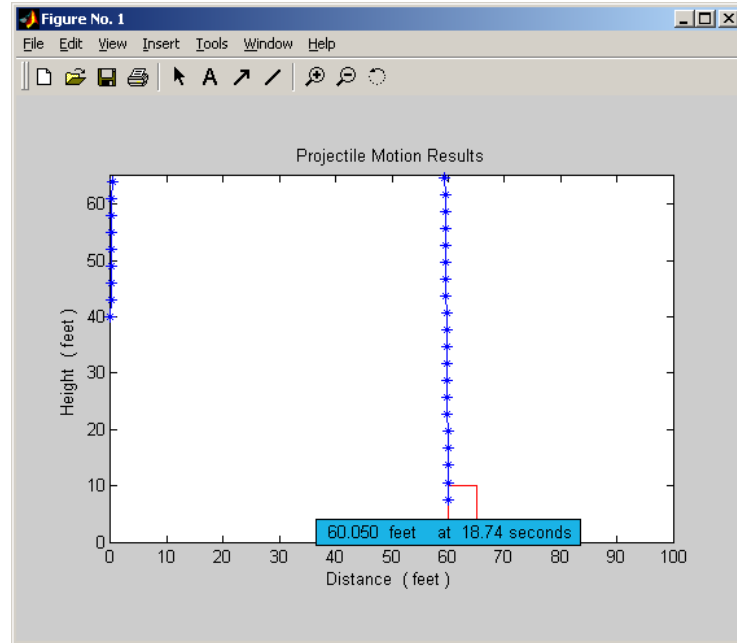


Figure 4.32 Hibbeler Homework 12-86 “High Arc” solution, with increased launch angle precision

Table 4.27 Hibbeler Homework 12-86 Time Increment and Angle Precision Results

Time Increment (seconds)	Angle (degrees)	X position (feet)	Percent difference from Hibbeler
Hibbeler	89.4	60	-
1	334	Through target	-
0.1	334	Through target	-
0.01	334	62.017	3.3617%
0.001	334	60.129	0.2150%
1	89.4	59.689	0.5183%
0.1	89.4	59.061	1.5650%
0.01	89.4	58.872	1.8800%
0.001	89.4	58.847	1.9217%
1	89.388	60.883	1.4717%
0.1	89.388	60.242	0.4033%
0.01	89.388	60.050	0.0833%
0.001	89.388	60.024	0.0400%

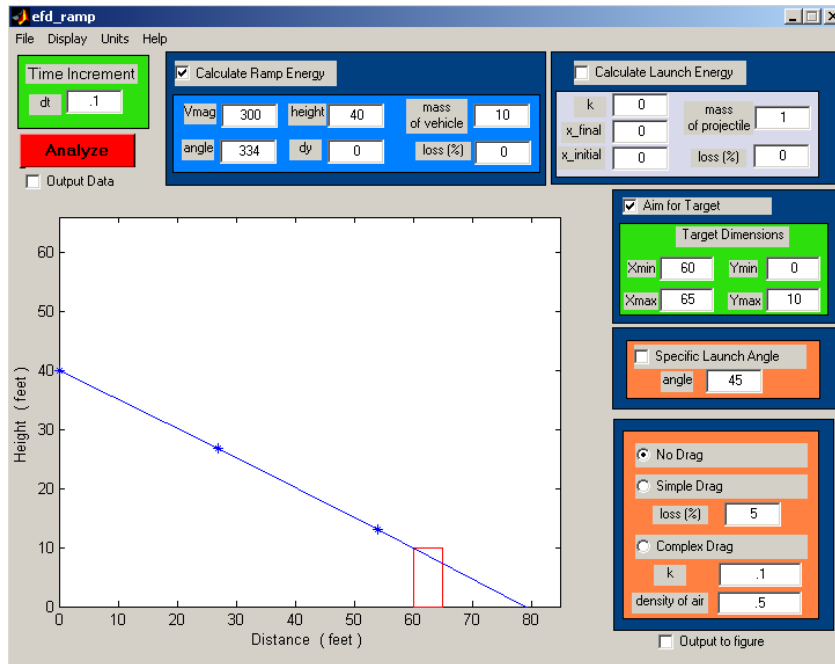


Figure 4.33 Hibbeler Homework 12-86 Direct Path missing target

4.2.4 Boresi / Schmidt Example 14-7, Drag Verification

A batter hits a baseball at a height of 4 feet above the ground, (Figure 4.34). The ball leaves the bat with a speed of $V_o = 90 \text{ mi/hr}$ (132 ft/s), at an angle of 40° relative to the ground. The mass of the ball is $m = 0.009931 \text{ slugs}$, and the radius of the ball is $r = 1.44 \text{ in}$. The mass density of the air is $0.002328 \text{ slug/ft}^3$. Determine the horizontal distance R traveled by the ball before it strikes the ground. First, neglect drag. Then, assume a drag coefficient of $k = 0.001 \text{ lb-s/ft}$. The data is shown in Table 4.28.

The time increment of 0.001 seconds is used because time becomes less of a factor in the consistency of the path results. In the Ramp Energy Menu, The initial velocity, V_{mag} , is set to 132 feet/sec, the angle is set to 40° CCW of the x-axis,

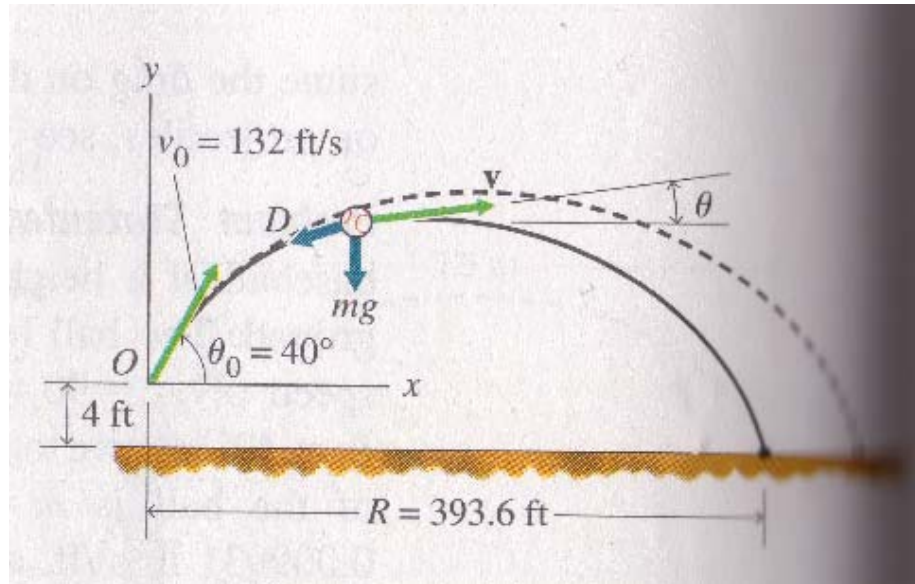


Figure 4.34 Boresi / Schmidt Example 14-7 Figure

Table 4.28 Boresi / Schmidt Example 14-7 data entered into application

Variable	Value	Menu
dt	0.001	Time Increment
Vmag	132	Ramp Energy
Angle (CCW from x-axis)	40	Ramp Energy
height	4	Ramp Energy
dy	0	Ramp Energy
mass	0.009931	Ramp Energy
k	0.001	Drag
units	English	Toolbar

The image shows a software interface for calculating drag. It is divided into three main sections:

- Top Left (Green Box):** Contains a "Time Increment" label, a "dt" input field with the value ".001", a red "Analyze" button, and an "Output Data" checkbox.
- Top Right (Blue Box):** Contains a checked "Calculate Ramp Energy" checkbox. Below it are input fields for "Vmag" (132), "height" (4), "mass of vehicle" (.00993), "angle" (40), "dy" (0), and "loss (%)" (0).
- Bottom (Orange Box):** Contains three radio button options: "No Drag", "Simple Drag", and "Complex Drag". The "Complex Drag" option is selected. Below the "Complex Drag" option is a "k" input field with the value ".001".

Figure 4.35 Boresci / Schmidt Drag Example 14-7 User Entry

the initial height is 4 feet, the mass is 0.009931 slugs, and there is no ramp, which results in a dy of 0. To account for drag, the coefficient of drag value k is 0.001 lb-s/foot and the “Complex Drag” option must be selected. The user entry will match Figure 4.35.

The results are shown in Table 4.29 and Figure 4.36. Once again, the application calculates a very precise answer when neglecting drag, but some error when considering drag. The results include a simple drag calculation.

An initial velocity loss of 27% was chosen because the effect of the complex drag distance (393 feet) resulted in 27% less distance when compared to the neglected drag distance (537 feet). The simple drag calculation is to be used as a quick estimation if the student has no coefficient of drag for a complex calculation. For this example, the assumption of 27% loss in initial velocity was within 0.3% of the length, but 8% error for the time calculation. Obviously, this simple drag calculation is not the correct method for

Table 4.29 Boresi / Schmidt Example 14-7 Results

Variable	Application Answer	Text Answer	Percent diff
R no drag	537.623 feet	537.6 feet	0.004278%
Time no drag	5.3168 seconds	5.3168 seconds	0.0000%
R with complex drag	392.824 feet	393.6 feet	0.1971%
Time with complex drag	4.9280 seconds	4.9302 seconds	0.04462%
R with simple 27% drag	392.480 feet	393.6 feet	0.2845%
Time with simple 27% drag	5.318 seconds	4.9302 seconds	7.866%

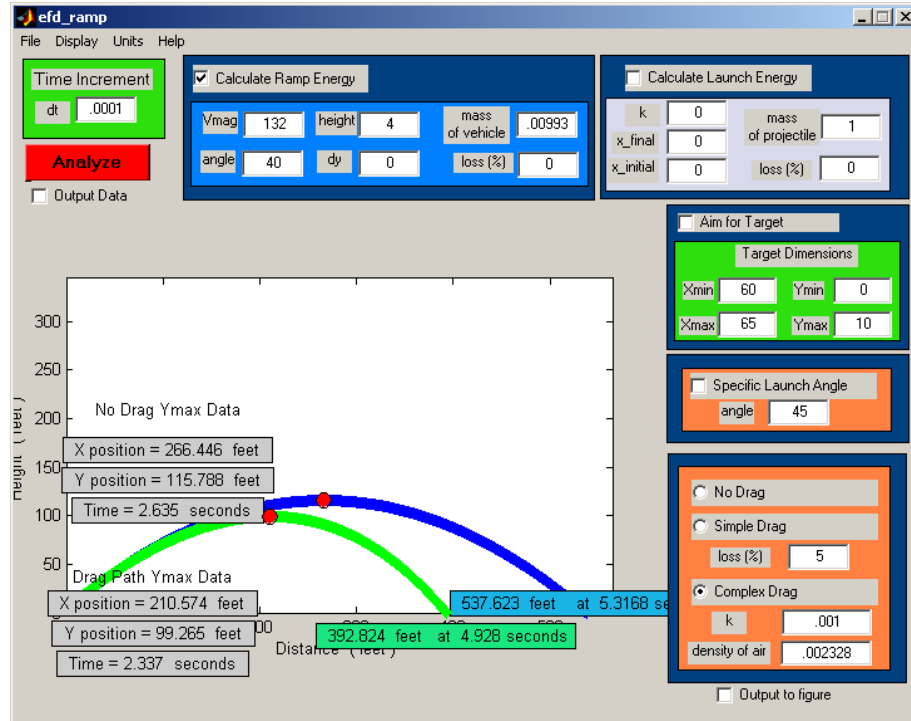


Figure 4.36 Boresi / Schmidt Example 14-7 Output

Table 4.30 Boreshi / Schmidt Example 14-7 Application, Excel, and Text Answer

Variable	Application Answer	Excel Answer	Text Answer
R with complex drag	392.824 feet	392.824 feet	393.6 feet
Time with complex drag	4.9280 seconds	4.9280 seconds	4.9302 seconds

calculating the drag path. However, given the crudeness of the simple drag assumption, the application calculated a reasonable flight path.

The application calculated the complex drag path within 0.2% of the correct length and 0.04% of the correct time. The same drag calculations were entered into an Excel spreadsheet, which calculated the same values as the application, (see Table 4.30). One possible explanation is the precision of the coefficients given with the problem, similar to the Hibbeler Homework 12-86 “High Arc” solution. Only this time, more precise coefficients cannot be calculated for use in the application.

The “Complex Drag” calculation is accurate within a necessary precision for use in the *engage* dynamics team projects. The “Simple Drag” calculation is not accurate, but a good estimation tool for preliminary analyses.

4.3 Swing Energy Application Verification

Only one verification problem is needed since the basic projectile motion and drag calculations were verified in section 4.2. Homework 14-31 demonstrates the application’s ability to calculate the proper swing exit velocity.

4.3.1 Homework 14-31

Marbles having a mass of 5 g fall from rest at point A through the glass tube and accumulate in the can at C, (Figure 4.37). Determine the placement R of the can from the end of the tube and the speed at which the marbles fall into the can. Neglect the size of the can. The data that needs to be entered into the application is shown in Table 4.31.

In the Properties Menu, the marbles have a mass of 5 g (0.005 kg) and start from rest, so the initial velocity V_t is 0. The time increment was chosen by starting at 0.1 seconds and decreasing the by a factor of 10 until the calculations remain constant. In the Swing Energy Menu, to simulate the center of the swinging motion , the center point of the swinging is at $X = 0$ and $Y = 3$ meters with a swing radius of 1 meter. The start angle is 180 degrees and swings to 270 degrees. At the release angle, the marble launches tangent to the release angle. In the Particle Release Menu, the “With Release” option must be selected. Also, the *projectile mass* value of 0.005 kg must be entered. The *projectile mass* is used to calculate the launch velocity from the kinetic energy from the falling vehicle. In this homework, the vehicle and projectile are the same object, so the mass needs to match in both data entries. The data should match Figure 4.38.

The results of the calculation are shown in Figure 4.39 and Table 4.32. The application calculates an accurate answer relative to the precision of the Hibbeler answers.

The application properly calculates an object swinging at a radius and launching tangent to the release angle.

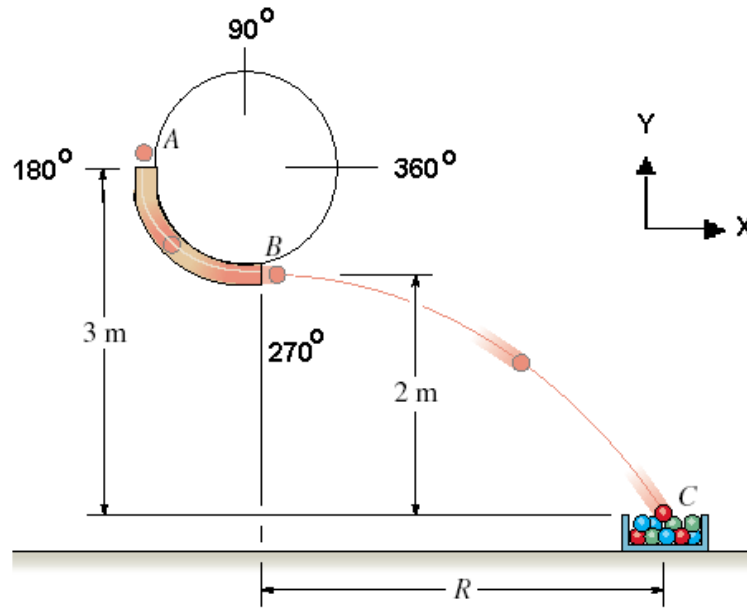


Figure 4.37 Hibbeler Homework 14-31 Figure

Table 4.31 Homework 14-31 data for application

Variable	Value	Menu
dt	0.001	Properties
Vt	0	Properties
Vehicle mass	0.005	Properties
X	0	Swing Energy
Y	3	Swing Energy
radius	1	Swing Energy
Start angle	180	Swing Energy
Release angle	270	Swing Energy
Projectile mass	0.005	Swing Energy
units	Metric	Toolbar

Properties

dt0.001

Vt0

vehicle mass0.005

☒ Calculate Swing Energy

Swing Coordinates

X0start angle180

Y3release angle270

radius1loss (%)0

Particle Release Angle

☒ With release

☐ Offset0

☐ Specify0

projectile mass0.005

Figure 4.38 Data for Hibbeler Homework 14-31

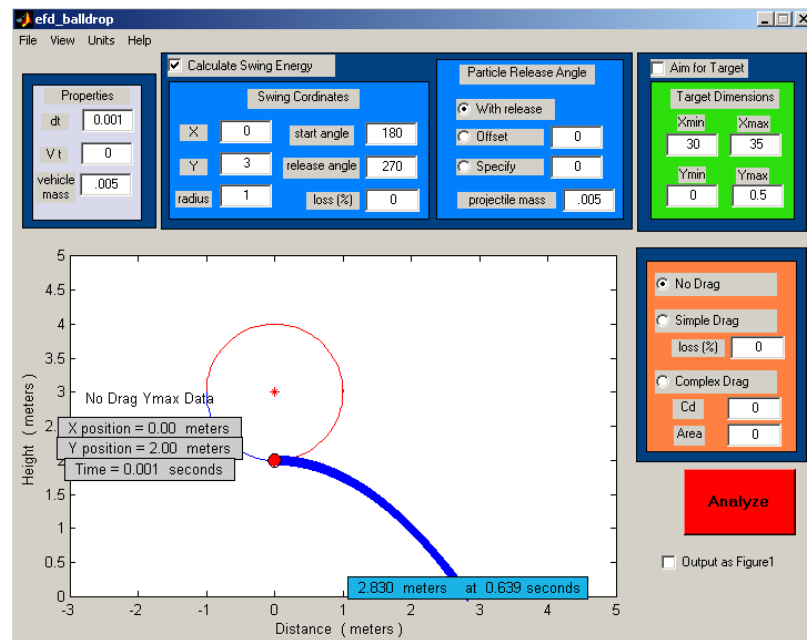


Figure 4.39 Hibbeler Homework 14-31 application results

Table 4.32 Hibbeler Homework 14-31 application results

Variable	Application Answer	Text Answer
Length R	2.830 meters	2.83 meters
Velocity at B	$V_x = 4.429 \text{ m/s}$	$V_x = 4.429 \text{ m/s}$
Velocity at C	$V_x = 4.429$ $V_y = -6.269$ $V_{\text{mag}} = 7.676$	$V_{\text{mag}} = 7.67$

Chapter 5 - Conclusions and Recommendations

5.1 Conclusions

Three analysis applications for use with the *engage* program's team projects were created. The applications were programmed in Matlab, utilize a simple Graphical User Interface (GUI), include corresponding help files and walkthrough tutorials, and can be used to analyze the typical statics and dynamics team projects assigned in EF 102.

These applications provide a quick and simple analysis of the type of designs related to the *engage* team projects. The students will be able to enter their design information into the appropriate application and obtain an answer that is accurate to the precision of the input data. Thus, the applications allow the students to accurately analyze their designs and determine if the calculations have the intended result. Also, they can make hypothetical modifications to the and asses proposed design improvements.

For each application, all user input menus are part of the main GUI window. This approach simplifies the data input task and provides a visual feedback as they build their model. The help files and tutorials serve as a guide to teach the user how to use the application and to familiarize themselves with the analysis capabilities.

Matlab is the clear choice as the programming language for two reasons. First, it is the current programming language taught in the *engage* program. Therefore, the students are given a working example of complex Matlab code. Second, it is versatile

relating to matrix storage, matrix computations, plotting capabilities, and numeric analysis functions.

5.2 Recommendations

While each application is capable of analyzing statics or dynamics team projects that have typically been assigned in the past, there are some modifications outside of the initial project scope that would make the applications either more applicable to the team projects by adding additional analysis or more entertaining for the students by providing a game inside the projectile motion applications.

5.2.1 Truss Solver Application Modifications

There are two modifications to the truss solver application that would enhance its analysis capability. The first modification is to add the capability to easily analyze a single load moving from joint to joint. The moving load would simulate a student walking over the bridge. The second modification is to add the ability to compare a member's strength versus the member's applied load. With this added data, the application could determine and report if a member will fail.

5.2.2 Projectile Motion Application Modifications

There are three modifications to the projectile motion applications that may be desired by the user. First, the spring energy could be solved iteratively to account for a

non-linear spring as the projectile launches. Second, the projectile motion applications can be modified to include a randomly positioned target game. Third, the swing energy can be modified to account for trebuchet style launching.

5.2.3 Implementation Suggestions

The applications must be introduced to the students in a way that is coordinated within the *engage* class structure. When it is time for the students to use the applications for their team project, they will have been taught the necessary mechanics concepts from the various components of *engage*. Therefore, after the application and the walkthrough tutorials are introduced to the students in Analysis and Skills, the students should be able to use the available information to apply the applications to their team project.

The file format can be as Matlab files or as executable files. To distribute as Matlab files, the user needs to have the m-file and figure file associated with the application. However, a set of executable files can also be distributed. In either case, the help files need to be in the same directory as the respective applications. As an executable, the user would not need Matlab installed on his computer to run the applications. However, as Matlab files, the user can view the code and follow the calculations.

REFERENCES

References

- [1] Parsons, J. R., The Engage Program: Implementing and Assessing a New First Year Experience at the University of Tennessee. *Journal for Engineering Education*: 441-446, October 2002.
- [2] *Matlab* – Numerical Mathematical Software Package specializes in matrix computations, online web pages at <http://www.mathworks.com>
- [3] *West Point Bridge Designer* – Bridge Analysis Software with a focus on minimizing the cost of support beams, online web pages at <http://bridgecontest.usma.edu/>
- [4] Shield, T. W., *A simple 2D truss solver program*, University of Minnesota, online web pages at <http://www.aem.umn.edu/people/faculty/shield/software/truss/>
- [5] *Cosmos* – Finite Element Software, online web pages at <http://www.cosmos.com>
- [6] *Ansys* – Finite Element Software, online web pages at <http://www.ansys.com>
- [7] *Simple Web Based Projectile Motion Program* – Provided by the University of Oregon Physics Department, online web pages at <http://zebu.uoregon.edu/nsf/cannon.html>
- [8] *Simple Excel File Projectile Motion Program* – George Mason University, Physics and Astronomy Department, online web pages at <http://www.physics.gmu.edu/~jevans/phys251/Topics/ScientificComputing/spreadSheets.html>
- [9] Pionke, C. D., Trusses, Matlab Method of Joints. *EF 102 Lecture 2-6*: Spring 2003.
- [10] *Mechanical Desktop* – 3D Solid Modeling and 2D CAD tool, online web pages at <http://usa.autodesk.com/>
- [11] Boresi, A.P. and Schmidt, R.J., **Engineering Mechanics Dynamics**, Brooks/Cole Publishing Company, 2001.
- [12] *PETSc* – Portable Extensive Toolkit for Scientific computing, online web pages at <http://acts.nersc.gov/petsc/main.html>
- [13] *DirectX* – Microsoft 3D graphics engine, online web pages available at <http://www.microsoft.com/directx>
- [14] *OpenGL* – Open Source 3D graphics engine, online web pages available at <http://www.opengl.org/>

[15] Hibbeler, R.C., **Engineering Mechanics Statics 9th Edition**, Prentice Hall Publishing Company, 2001.

[16] Hibbeler, R.C., **Engineering Mechanics Dynamics 9th Edition**, Prentice Hall Publishing Company, 2001.

APPENDICES

APPENDIX A

EF 102 Spring 2003 Calendar and Team Projects

EF 102 - Module 1 Topics and Schedule

- Click on lecture titles to view the lecture outline.
- Click on the HW number to view the solution (posted shortly after all sections have turned it in)
- Items in **red** are updates and corrections to the original posting of this page
- [Semester Calendar](#) | [Module 1](#) | [Module 2](#) | [Module 3](#) | [Module 4](#) | [Module 5](#)

Mon	Tue	Wed	Thu	Fri																														
Jan 13  Vectors Course Overview Read 1.1-1.5, 2.1-2.6	Jan 14	Jan 15 Lecture 1-2   Position Vectors Forces along a line Dot Product Read 2.7 - 2.9	Jan 16	Jan 17 Lecture 1-3   Moment of a Force Read 4.1 - 4.2																														
	PS 1-1		PS 1-2																															
	<table><tr><td></td><td>Suggested</td><td>HW</td></tr><tr><td rowspan="4">Handout</td><td>2-19</td><td>2-32</td></tr><tr><td>2-31</td><td>2-48</td></tr><tr><td>2-40</td><td>2-54</td></tr><tr><td>2-55</td><td></td></tr></table>			Suggested	HW	Handout	2-19	2-32	2-31	2-48	2-40	2-54	2-55		<table><tr><td>In-class</td><td>Suggested</td><td>HW</td></tr><tr><td>2-82</td><td>2-85</td><td>2-83</td></tr><tr><td>2-97</td><td>2-89</td><td>2-96</td></tr><tr><td>2-121</td><td>2-90</td><td>2-112</td></tr><tr><td></td><td>2-114</td><td>AS-1</td></tr><tr><td></td><td>2-125</td><td>assignment</td></tr></table>		In-class	Suggested	HW	2-82	2-85	2-83	2-97	2-89	2-96	2-121	2-90	2-112		2-114	AS-1		2-125	assignment
		Suggested	HW																															
	Handout	2-19	2-32																															
2-31		2-48																																
2-40		2-54																																
2-55																																		
In-class	Suggested	HW																																
2-82	2-85	2-83																																
2-97	2-89	2-96																																
2-121	2-90	2-112																																
	2-114	AS-1																																
	2-125	assignment																																
	Team 1		PH 1																															
Mon	Tue	Wed	Thu	Fri																														
Jan 20 MLK Day	Jan 21	Jan 22 Lecture 1-4   Vector Cross Product Moment of a Force 3-D Moments Read 4.3 -	Jan 23	Jan 24 Lecture 1-5   Moment of a Couple Equivalent Systems Read 4.6 - 4.9																														

		4.5		
	PS 1-3		PS 1-4	
	In-class	Suggested	HW Due PS 1-5	In-class Suggested HW Due PS 1-6
	4-9 4-15 4-30	4-8 4-10 4-11 4-22	4-13 4-14 4-18* * Use Matlab to generate the plot.	4-27 4-55 4-25 4-29 4-42 4-47 4-57 4-59 4-7* 4-39 4-58 AS-2 assignment * Use cross product method
	Team 2		PH 2	
Mon	Tue	Wed	Thu	Fri
Jan 27 Lecture 1-6  Completed m.file used in class Matlab Vector Operations Moments	Jan 28	Jan 29 Lecture 1-7  Review/Perspective Sample 2000   Sample 2001  Sample 2002  Matlab and TI-85 solution to barndoor problem	Jan 30	Jan 31 Exam 1 
	PS 1-5		PS 1-6	
	In-class	Suggested	HW Due PS 1-6	In-class Suggested HW
	4-55* 4-72 4-101 4-109 * problem from PS 1-4	4-73 4-76 4-106 4-113	4-70 4-105 4-123	A&S Power Point Slides Example function None AS-3
	Team 3 (PH 2)		PH 3	

EF 102 - Module 2 Topics and Schedule

- Click on lecture titles to view the lecture outline.
- Click on the HW number to view the solution (posted shortly after all sections have turned it in)
- Items in **red** are updates and corrections to the original posting of this page
- [Semester Calendar](#) | [Module 1](#) | [Module 2](#) | [Module 3](#) | [Module 4](#) | [Module 5](#)

Mon	Tue	Wed	Thu	Fri																											
Feb 3 Lecture 2-1   Equilibrium, 2D FBD Read 3.1 - 3.4, 5.1 - 5.3	Feb 4	Feb 5 Lecture 2-2   Equilibrium, 2D Beams Simply Pulleys	Feb 6	Feb 7 Lecture 2-3   Equilibrium, 3D Constraints / Reactions Read 5.4 - 5.7																											
	PS 2-1		PS 2-2																												
	<table><tr><th>In-class</th><th>Suggested</th><th>HW</th></tr><tr><td>3-2</td><td>3-5</td><td>3-1</td></tr><tr><td>3-8</td><td>3-21</td><td>3-15</td></tr><tr><td>5-2</td><td>5-5</td><td>5-4</td></tr><tr><td></td><td></td><td>AS-3</td></tr></table>		In-class	Suggested	HW	3-2	3-5	3-1	3-8	3-21	3-15	5-2	5-5	5-4			AS-3	<table><tr><th>In-class</th><th>Suggested</th><th>HW</th></tr><tr><td>5-14</td><td>5-22</td><td>5-27</td></tr><tr><td>5-20</td><td>5-23</td><td>5-28</td></tr><tr><td>5-29</td><td>5-26</td><td>5-24</td></tr></table>		In-class	Suggested	HW	5-14	5-22	5-27	5-20	5-23	5-28	5-29	5-26	5-24
	In-class	Suggested	HW																												
3-2	3-5	3-1																													
3-8	3-21	3-15																													
5-2	5-5	5-4																													
		AS-3																													
In-class	Suggested	HW																													
5-14	5-22	5-27																													
5-20	5-23	5-28																													
5-29	5-26	5-24																													
	Team 4		PH 4																												

Mon	Tue	Wed	Thu	Fri
Feb 10 Lecture 2-4   Plane Trusses Method of Joints Read 6.1 - 6.3	Feb 11	Feb 12 Lecture 2-5   Method of Sections Space Trusses Read 6.4 - 6.5	Feb 13	Feb 14 Lecture 2-6   Completed m.file used in class Trusses – Matlab Method of Joints

				PS 2-3			PS 2-4		
				In-class	Suggested	HW	In-class	Suggested	HW
				5-67	5-79	5-72	6-31	6-33	6-37
				6-5	6-9	6-7	6-43	6-46	6-44
					6-12		6-45	6-47	6-49
					6-16		6-19*		AS-4
				Use method of joints.			except 6-19. Solve for each unknown without using the other unknowns.		
				Team 5			PH 5		

Mon	Tue	Wed	Thu	Fri
Feb 17 Lecture 2-7  Multi-Force Members Frames & Machines Read 6.6	Feb 18	Feb 19 Lecture 2-8  Frames & Machines Pulley Systems	Feb 20	Feb 21 Lecture 2-9  Review/Perspective Review Exams

				PS 2-5			PS 2-6		
				In-class	Suggested	HW	In-class	Suggested	HW
				6-80	6-88	6-86 pt D	A&S	6-68	6-67
				6-95	6-89	is fixed	Power Point	6-69	AS-
				6-128	6-90	6-96	Slides		5
							truss2d m.files		
				Team 6			PH 6		

Mon	Tue	Wed	Thu	Fri
Feb 24 Exam 2 Exam Results	Feb 25	Feb 26		

	PS 2-7			
	In-class	Suggested	HW	
	A&S Read ASR 11.1 - 11.2 Power Point Slides m.file used in class	na	AS-6 This will be assigned Friday, Feb. 28.	
	Team 7			

EF 102 - Module 3 Topics and Schedule

- Click on lecture titles to view the lecture outline.
- Click on the HW number to view the solution (posted shortly after all sections have turned it in)
- Items in **red** are updates and corrections to the original posting of this page
- [Semester Calendar](#) | [Module 1](#) | [Module 2](#) | [Module 3](#) | [Module 4](#) | [Module 5](#)

Mon	Tue	Wed	Thu	Fri	
Feb 24	Feb 25	Feb 26 Lecture 3-1   Kinematics Review Graphs Constant Acceleration Read 12.1-12.3	Feb 27	Feb 28 Lecture 3-2   Matlab Application: s-t, v-t, a-t diagrams Read ASR 11.3 - 11.5 <u>Completed</u> <u>m.file used in</u> <u>class</u> <u>Final s-v-a plot</u> <u>used in class</u>	
			PS 3-1		
			In-class	Suggested	HW
			12-15 12-19 <u>Matlab</u> <u>solution</u> 12-49/50 (numerically)	12-10 12-49 12-53	<u>12-</u> <u>23</u> <u>12-</u> <u>31</u> <u>12-</u> <u>57</u>
	Team 7	<u>PH 7</u>			
Mon	Tue	Wed	Thu	Fri	
Mar 3 Lecture 3-3   Projectile Motion Read 12.4-	Mar 4	Mar 5 Lecture 3-4   Normal and Tangential Components	Mar 6	Mar 7 Lecture 3-5   Constrained Motion Read 12.9	

12.6		Read 12.7		
	PS 3-2			PS 3-3
	In-class	Suggested	HW	In-class Suggested HW
	12-83	12-81	12-80	12-100 12-107 12-118 12-102 12-104 12-119 (magnitude & direction)
	12-85	12-87	12-84	
	12-91	12-90	12-92	
	Team 8 - Bridge Testing			PH 8
Mon	Tue	Wed	Thu	Fri
Mar 10 Lecture 3-6  Matlab Application: Projectile Motion Files used in class	Mar 11	Mar 12 Lecture 3-7  Review/Perspective Review Exams	Mar 13	Mar 14 Exam 3 Exam Results
	PS 3-4			PS 3-5
	In-class	Suggested	HW	In-class Suggested HW
	12-173	12-174	12-	TBA
	12-179	12-175	172	
	12-185	12-178	12-	
	Example 12-24		181	
			12-	
			183	
	Team 9			PH 9

EF 102 - Module 4 Topics and Schedule

- Click on lecture titles to view the lecture outline.
- Click on the HW number to view the solution (posted shortly after all sections have turned it in)
- Items in **red** are updates and corrections to the original posting of this page
- [Semester Calendar](#) | [Module 1](#) | [Module 2](#) | [Module 3](#) | [Module 4](#) | [Module 5](#)

Mon	Tue	Wed	Thu	Fri																														
Mar 24 Lecture 4-1   Kinetics Read 13.1-13.4 AS-7 assignment	Mar 25	Mar 26 Lecture 4-2   Connected Bodies	Mar 27	Mar 28 Lecture 4-3   Matlab Applications Problem 13-12 m.file, trap method Problem 13-12 m.file, poly method																														
	PS 4-1 <table><tr><th>In-class</th><th>Suggested</th><th>HW</th></tr><tr><td>13-1</td><td>13-5</td><td>13-8</td></tr><tr><td>13-3</td><td>13-14</td><td>13-13</td></tr><tr><td>13-12</td><td>13-28</td><td>13-35</td></tr><tr><td>13-31</td><td></td><td></td></tr></table>		In-class	Suggested	HW	13-1	13-5	13-8	13-3	13-14	13-13	13-12	13-28	13-35	13-31			PS 4-2 <table><tr><th>In-class</th><th>Suggested</th><th>HW</th></tr><tr><td>13-6</td><td>13-20</td><td>13-27</td></tr><tr><td>13-19</td><td>13-24</td><td>13-30</td></tr><tr><td>13-19 (w/ 15 lb weight)</td><td></td><td>13-36</td></tr><tr><td>13-25</td><td></td><td></td></tr></table>		In-class	Suggested	HW	13-6	13-20	13-27	13-19	13-24	13-30	13-19 (w/ 15 lb weight)		13-36	13-25		
In-class	Suggested	HW																																
13-1	13-5	13-8																																
13-3	13-14	13-13																																
13-12	13-28	13-35																																
13-31																																		
In-class	Suggested	HW																																
13-6	13-20	13-27																																
13-19	13-24	13-30																																
13-19 (w/ 15 lb weight)		13-36																																
13-25																																		
	Team 10		PH 10																															

Mon	Tue	Wed	Thu	Fri
Mar 31 Lecture 4-4   Kinetic Friction	Apr 1	Apr 2 Lecture 4-5   Static Friction Read 8.1-8.2	Apr 3	Apr 4 Lecture 4-6   Normal-Tangential Read: 13.5 AS 8 assignment

PS 4-3 (These problems will be switched to the Thu/Fri problem session) <table><tr><th>In-class</th><th>Suggested</th><th>HW</th></tr><tr><td>8-8</td><td>8-14</td><td>8-15</td></tr><tr><td>8-17</td><td>8-34</td><td>8-24</td></tr><tr><td>8-18</td><td>8-37</td><td>8-44</td></tr><tr><td>8-26</td><td></td><td></td></tr></table>		In-class	Suggested	HW	8-8	8-14	8-15	8-17	8-34	8-24	8-18	8-37	8-44	8-26			PS 4-4 (These problems will be switched to the Tue/Wed problem session) <table><tr><th>In-class</th><th>Suggested</th><th>HW</th></tr><tr><td>13-9</td><td>13-16</td><td>13-18</td></tr><tr><td>13-11</td><td>13-23</td><td>13-21</td></tr><tr><td>13-33</td><td>13-43</td><td>13-22</td></tr></table>		In-class	Suggested	HW	13-9	13-16	13-18	13-11	13-23	13-21	13-33	13-43	13-22
In-class	Suggested	HW																												
8-8	8-14	8-15																												
8-17	8-34	8-24																												
8-18	8-37	8-44																												
8-26																														
In-class	Suggested	HW																												
13-9	13-16	13-18																												
13-11	13-23	13-21																												
13-33	13-43	13-22																												
Team 11		PH 11																												

Mon	Tue	Wed	Thu	Fri
Apr 7 Lecture 4-7  Work-Energy Read 14.1-14.3	Apr 8	Apr 9 Lecture 4-8  Conservation of Energy Read 14.5, 14.6	Apr 10	Apr 11 Lecture 4-9  Review/Perspective Bungee jump plots Review Exams Sunday Review Problems Sunday Review Screen Shots

PS 4-5 <table><tr><th>In-class</th><th>Suggested</th><th>HW</th></tr><tr><td>13-53</td><td>13-55</td><td>13-54</td></tr><tr><td>13-70</td><td>13-57</td><td>13-56</td></tr><tr><td>13-61</td><td>13-59</td><td>13-60</td></tr><tr><td>13-62</td><td></td><td></td></tr></table>		In-class	Suggested	HW	13-53	13-55	13-54	13-70	13-57	13-56	13-61	13-59	13-60	13-62			PS 4-6 <table><tr><th>In-class</th><th>Suggested</th><th>HW</th></tr><tr><td>14-6</td><td>14-28</td><td>14-18</td></tr><tr><td>14-14</td><td>14-78</td><td>14-19</td></tr><tr><td>14-27</td><td>14-93</td><td>14-20</td></tr></table>		In-class	Suggested	HW	14-6	14-28	14-18	14-14	14-78	14-19	14-27	14-93	14-20
In-class	Suggested	HW																												
13-53	13-55	13-54																												
13-70	13-57	13-56																												
13-61	13-59	13-60																												
13-62																														
In-class	Suggested	HW																												
14-6	14-28	14-18																												
14-14	14-78	14-19																												
14-27	14-93	14-20																												
Team 12		PH 12																												

Mon	Tue	Wed	Thu	Fri
Apr 14 Exam 4 Exam Results	Apr 15 Start of Module 5	Apr 16	Apr 17	Apr 18

PS 5-1 A&S in computer lab		
Team 13		

EF 102 - Module 5 Topics and Schedule

- Click on lecture titles to view the lecture outline.
- Click on the HW number to view the solution (posted shortly after all sections have turned it in)
- Items in **red** are updates and corrections to the original posting of this page
- [Semester Calendar](#) | [Module 1](#) | [Module 2](#) | [Module 3](#) | [Module 4](#) | [Module 5](#)

Mon	Tue	Wed	Thu	Fri												
Apr 14 Exam 4 Exam Results	Apr 15	Apr 16 Lecture 5-1 _B  Linear Momentum Read 15.1-15.3 AS-9 assignment	Apr 17	Apr 18 Spring Recess												
	PS 5-1 A&S in computer lab		PS 5-2 <table><tr><th>In-class</th><th>Suggested</th><th>HW</th></tr><tr><td>15/17</td><td>15/18</td><td>15-16</td></tr><tr><td>15/33</td><td>15/47</td><td>15-32</td></tr><tr><td>15/49</td><td></td><td>15-50</td></tr></table>		In-class	Suggested	HW	15/17	15/18	15-16	15/33	15/47	15-32	15/49		15-50
In-class	Suggested	HW														
15/17	15/18	15-16														
15/33	15/47	15-32														
15/49		15-50														
	Team 13		PH 13													

Mon	Tue	Wed	Thu	Fri																								
Apr 21 Lecture 5-2 _P  Central Impact Read 15.4 Link to simulation files	Apr 22	Apr 23 Lecture 5-3 _P  Oblique Impact	Apr 24	Apr 25 Lecture 5-4 _B   Review Problems																								
PS 5-2 (WE,WF) PH 13 (WE,WF)	PS 5-3 <table><tr><th>In-class</th><th>Suggested</th><th>HW</th></tr><tr><td>15/57</td><td>15/58</td><td>15-56</td></tr><tr><td>15/69</td><td>15/67</td><td>15-64</td></tr><tr><td>15/75</td><td>15/73</td><td></td></tr></table>		In-class	Suggested	HW	15/57	15/58	15-56	15/69	15/67	15-64	15/75	15/73		PS 5-4 <table><tr><th>In-class</th><th>Suggested</th><th>HW</th></tr><tr><td>15/83</td><td>15/66</td><td>15-77</td></tr><tr><td>15/87</td><td>15/85</td><td>15-80</td></tr><tr><td>15/81</td><td></td><td></td></tr></table>		In-class	Suggested	HW	15/83	15/66	15-77	15/87	15/85	15-80	15/81		
In-class	Suggested	HW																										
15/57	15/58	15-56																										
15/69	15/67	15-64																										
15/75	15/73																											
In-class	Suggested	HW																										
15/83	15/66	15-77																										
15/87	15/85	15-80																										
15/81																												
	PH 14		Team 14 - Project Testing																									

Mon	Tue	Wed	Thu	Fri

Apr 28 Lecture 5-5 _P  A&S Review	Apr 29	Apr 30 Lecture 5-6 _B  Semester Wrap-up	May 1 Final Exam Review AMB 210 10am-noon Problems	May 2
	PS 5-5 A&S Exam in computer lab			
	Team 15			

EF 102 Statics Team Project

EF 102 Design Project 1, Spring 2003

Preliminary Assignments - It's a Bridge!

Engineers are responsible for the integrity and safety of their designs, a fact that causes sleepless nights, ethical dilemmas, and all sorts of other fun aspects of being a professional. Your diabolical instructors would like to give you a flavor of this concept by asking you to design a bridge where the test load(s) will be you, the designers of the bridge.

The scale of your construction will be similar to the bridges shown on the Brunel video, but unlike the students shown, you will have the time and resources to go through the steps of the design process. Schedule and details on specifications and materials will be furnished next week.

We are going to require you to keep a record of your design work for this class in a notebook (English Composition Book or equivalent). Now is the time to purchase this and begin recording your activity. As a first assignment with your new team, your instructors ask that you spend about 15 to 20 minutes letting each team member give contact information and sharing a positive and a negative experience from his or her last semester's team. Each team should then put together a short "rule list" that will constitute how you agree to work together this semester. When you get your notebook, each team member should copy or paste a copy of this "team rule list" into his or her book.

Your second initial assignment is to gather information on bridge design that will be input to your idea generation for this project. Before you leave today each team member should have an information gathering assignment that is specific enough that each member can report to the team next week.

There is lots of material available - student design contests, texts on the subject (general description of bridge types only), personal observation and experience, Civil Engineering department (display cases in Perkins)... the objective is that when you see the specifications, the team have adequate background to generate ideas.

EF 102 Design Project 1, Spring 2003

Bridge over Trouble Gorge

Your team is about to encounter Trouble Gorge, a devilish obstacle that has swallowed many unprepared students. Your team's assignment is to construct a bridge that will span this gorge of unspeakable dangers and then use your device to get your team members safely across. Once a year, some unknown seismic activity causes the atrium of Estabrook Hall to part and the gorge appears. Along with the gorge, a band of trolls from the 4th sub-basement of Estabrook make their yearly appearance, to harass you with a series of odd demands on your interaction with their environment.

Your bridge must be constructed with a limited supply of materials purchased from the Troll store at exorbitant costs (the troll council is still bickering on how much profit to make, but will let you know shortly about prices). Besides a profit, the troll band would like some entertainment from its visitors, so they have made a series of wagers on your efforts. One faction thinks you will be lucky to get one team member across your bridge, so this has been set as a minimum requirement. Another faction thinks your bridges should support the whole team, and this has been set as your maximum test load. They have decided, for fairness, to evaluate your bridges on a weight supported per dollar spent basis.

Trouble Gorge:

The gorge is 2.14 m wide (1/100 scale of the great Clifton gorge) and the troll band have helpfully supplied a sketch posted in your work area containing many useful dimensions from their recent survey of their domain. They insist that for minimum environmental impact your structures only interact with certain areas of the gorge ledges shown on this sketch.

Troll Store Stock List:

- 1.5 X 1.5 cm wood stock
- twine
- cotter pins appropriate for "pin" connections

Troll supplied "walking board", laid on top of your structures, will be 1.6 m long and 0.2 m wide.

Trolls will make available materials to run preliminary strength tests and to build scale models (Popsicle stick versions). Building full-scale test bridges will not be economically viable.

Troll Rules:

On 2/11-12/03, each team will give a 5-minute oral preliminary report covering the first stages of their design process. At the team's choice, you may use overheads or a poster for your visual aid. What would be the most effective

method to present your material (and earn much-needed brownie points with Big Boss Troll)?

Cross the Gorge Day will be 3/4-5/03

Team Deliverables:

1. Team bridge ready for testing on the gorge. Concerned for his band, the benevolent BBT will conduct a safety inspection of your devices before his home is subjected to falling students.
2. A written report of your project must be submitted to the troll council on Cross the Gorge Day. This report should follow the standard format you have practiced, with a problem statement in your own words that completely describes the problem and constraints, a background section that demonstrates to the reader that you are familiar with what other people have done on similar problems and the basic principles of mechanics that are applicable, a description of the different ideas considered by your team, the concept selection process with any applicable test results, a complete description of your final design, and a conclusions and recommendation section. In keeping with the material that the trolls know you are studying this semester, it will be expected that your design is backed up with the appropriate calculations and this will be a major factor in their report grade. As always, sketches and diagrams are important parts of conveying key points. Use of your computer tools (Mechanical Desktop and Matlab), integrated into your report, will be favorably received.
3. As part of your reporting for this project, you are asked to summarize your project on a team web page. The web page should consist of a summary of the Bridge over Trouble Gorge project, a brief listing of individual team members' contributions to its completion, at least one image from Mechanical Desktop, and links to each team member's personal home page. The team web page should reside on the University server, and its URL "registered" with our server via your [personal EF page](#). Resources regarding the creation of simple web pages and details regarding the creation of personal web pages will be provided as part of an upcoming A&S assignment.

The troll council will assign approximately equal credit for 1) bridge construction and performance and 2) reporting requirements.

EF 102 Dynamics Team Project

EF 102 Project 2

The Estabrook Critter-Conker

As long time residents of Estabrook Hall know, assorted crawling creatures often visit us. Most of these critters are nocturnal, so it is not a major inconvenience, but they have been known to agitate groups of students working late at night on their projects.

For this project, your team will design, construct and demonstrate a device that will discourage our nightly visitors by hitting them with a projectile when they appear. Your instructors, always anxious to demonstrate your calculational prowess, knowledge of dynamics, and creative design skills, have arranged the demonstration as follows:

Each team will design and construct a device that will fit in a designated 2 ft by 2 ft by 2 ft starting space, will launch a regulation projectile (paintball), and be constructed from a provided kit of materials (tentative list below). The critter will be released and will travel a path perpendicular to the launching direction of your device at a distance of 20, 25 or 30 feet from your device. The critter starts at a distance of 5 feet from the launch path of your device and travels at a constant velocity but the magnitude of the velocity is variable (in the range 0.2 to 0.8 ft/sec). Your device must be activated at the same time as the critter is released, and must then automatically launch the projectile at the proper time to hit the critter as it traverses in front of your device. To add to the challenge, a 4-ft high wall is located at a distance of 10 feet in front of your device. The trajectory of the projectile must clear this wall to successfully conk the critter. A rough sketch is provided below, further details will be released shortly.

Critter speed and distance will be announced before each attempt. Each team's device will be tested against two combinations of distance and critter speed with a nominal adjustment and set-up time allowed between runs. Scoring will be based on total distance from the target for the two attempts. A team member cannot touch your device after the critter has been released. No part of your device can leave the designated launch area. The launch area cannot be damaged during your set-up or operation of your device (no drilling, fastening, nailing...).

Tentative Materials Kit:

- 1 sheet foam core
- 4 ft of "bridge" wood stock
- 4 ft of duct tape
- 6 ft of twine
- 2 yardsticks
- 1 #108 rubber band
- 1/2 pound of small weights

1 mouse trap
1 4 ft 1/4 inch dowel
6 paper clips
4 small nails
3 oz Elmer's glue
2 pipe cleaners
1 small spring
Other material by petition – see schedule below
Wall Critter Paths
Launch Area
Impact Areas
Critter Starting Boxes

Your initial efforts should be directed toward developing background material, generating alternative designs, and selecting promising concepts. The information you have been given is sufficient to begin the design process. Please remember that the distance to the critter and critter speed will remain unknown until the day of competition and sufficient prediction and adjustment capability must be planned for. A prediction procedure utilizing Matlab will be required.

Questions for your consideration:

What are different ways that you can “time” your release mechanism?

What energy sources are available (and what is their magnitude)?

What is the size and mass of your projectile?

Is air drag on your projectile important?

What variables do you need to control to predict a projectile path?

How can you adjust the flight path?

How can you make your device reliable and repeatable?

Schedule and Reporting Requirements:

March 11-12 Project Assignment

March 25-26 Idea Generation – Material Petitions Accepted

April 1-2 Work Period – Material List Finalized

April 8-9 Preliminary design reports. Each team must demonstrate a (partially constructed) device, which can potentially accomplish the competition tasks. This will be an informal oral report.

April 11-16 Written report rough draft review (mandatory).

Event day will be Tuesday-Wednesday, April 22-23.

On Thursday, April 24, by 5 p.m., each team will a) post a written report described below as a team webpage, b) turn in design notebooks and a resume for each team member bundled together.

Written Report: This is your last project report of the year and you should look at this as an opportunity to demonstrate what you have learned about integrating your team

efforts into an interesting and complete description of your project. The integration of analysis into your design process is a particular item your instructors are looking for. This project requires less team “construction” in an effort to provide you with the time you need to do a good job on this report. This report should follow the standard format you have practiced, with a problem statement in your own words that completely describes the problem and constraints, a background section that demonstrates to the reader that you are familiar with what other people have done on similar problems and the basic principles of mechanics that are applicable, a description of the different ideas considered by your team, the concept selection process with any applicable test results, a complete description of your final design and timing prediction method, and a conclusions and recommendation section. As always, sketches and diagrams are important parts of conveying key points.

EF 102 Project 2

The Estabrook Critter Conker

Your petitions have been reviewed, duly considered, laughed at and finally resolved into the

Updated (and final) Materials List:*

From before -

- 1 sheet foam core
- 4 ft of “bridge” wood stock
- 4 ft of duct tape
- 2 yardsticks
- 1 # 108 rubber band
- 1 mouse trap
- 1 4 ft 1/4 inch dowel
- 6 paper clips
- 4 small nails
- 2 pipe cleaners
- 1 small spring
- 1/2 pound of small weights

Modifications to old list –

“6 ft of twine” changed to “You may use up to 20 ft of string/twine/fishing line”

“3 oz Elmer’s Glue” changed to “You may use glue for joining purposes”

And two reminders –

“1/2 pound of small weights” are “supplied by team”, your EF 102 tools are not acceptable for this purpose

108 rubber bands can be cut to make smaller bands, we can furnish one replacement per team

New Items –

- 1 marble
- 1 clothes pin
- 1 cotter pin
- 2 ft PVC pipe

Other New Items Available If Needed In Your Design (ie, we don’t have enough for everyone) -

- Small funnel
- DC Motor and battery box
- Sand

Other Permissible items (supplied by team) –

Small quantities of water, sheet of engineering paper, small quantities of scotch tape, a party balloon, dixie cup

*We have limited replacement supplies for most items

APPENDIX B

Applications' Code

Appendix B.1

2D / 3D Truss Solver Code

EFD_TRUSS

```
function varargout = efd_truss(varargin)
% EFD_TRUSS M-file for efd_truss.fig
%   EFD_TRUSS, by itself, creates a new EFD_TRUSS or raises the existing
%   singleton*.
%
%   H = EFD_TRUSS returns the handle to a new EFD_TRUSS or the handle to
%   the existing singleton*.
%
%   EFD_TRUSS('CALLBACK',hObject,eventData,handles,...) calls the local
%   function named CALLBACK in EFD_TRUSS.M with the given input arguments.
%
%   EFD_TRUSS('Property','Value',...) creates a new EFD_TRUSS or raises the
%   existing singleton*. Starting from the left, property value pairs are
%   applied to the GUI before efd_truss_OpeningFunction gets called. An
%   unrecognized property name or invalid value makes property application
%   stop. All inputs are passed to efd_truss_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
%   instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help efd_truss

% Last Modified by GUIDE v2.5 15-Jul-2003 18:14:20

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',    mfilename, ...
    'gui_Singleton', gui_Singleton, ...
    'gui_OpeningFcn', @efd_truss_OpeningFcn, ...
    'gui_OutputFcn', @efd_truss_OutputFcn, ...
    'gui_LayoutFcn', [] , ...
    'gui_Callback', []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```

% --- Executes just before efd_truss is made visible.
function efd_truss_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)
% varargin    command line arguments to efd_truss (see VARARGIN)

%clear global

% Choose default command line output for efd_truss
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% This sets up the initial plot - only do when we are invisible
% so window can get raised using efd_truss.
if strcmp(get(hObject,'Visible'),'off')

    surf(peaks);

    axis equal
end

global w_joint w_member w_constraint w_force

set(handles.mod_add, 'Value', 1);
set(handles.user_2d, 'Value', 1);
set(handles.user_2dxz, 'Value', 1);

% UIWAIT makes efd_truss wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = efd_truss_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in user_analyze.
function user_analyze_Callback(hObject, eventdata, handles)
% hObject    handle to user_analyze (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

```

```

global w_joint w_member w_constraint w_force w_all_data w_reaction_force w_force_data

%1. build matrix from members / joints
%2. apply forces then constraints

[j_length b] = size(w_joint);
[m_length b] = size(w_member);

all_data = zeros(j_length * 3, m_length);

%build matrix

for i = 1:j_length

    %cycle through each joint and find connectivity
    for j = 1:m_length

        xyz1 = 0;
        xyz2 = 0;

        %if member uses joint i
        if (w_member(j,1) == i | w_member(j,2) == i)

            %member data
            j1 = w_member(j,1);
            j2 = w_member(j,2);

            if j1 == i

                xyz1 = w_joint(j1,:);
                xyz2 = w_joint(j2,:);

                other_j = j2;

            elseif j2 == i

                xyz1 = w_joint(j2,:);
                xyz2 = w_joint(j1,:);

                other_j = j1;

            end

            %get lengths
            all_length = xyz2 - xyz1;

            scalar_length = sqrt(sum( all_length.^2));

            % adjust to member columns

```

```

        % x portion
        all_data(i*3-2, j) = all_length(1) / scalar_length;
        % y portion
        all_data(i*3-1, j) = all_length(2) / scalar_length;
        % z portion
        all_data(i*3-0, j) = all_length(3) / scalar_length;

    end
    %elseif anything?

    %for loops
end
end

[a b] = size(all_data);

force_data = zeros(j_length * 3, 1);

%create force vector
[a b] = size(w_force);

for i = 1:a

    j1 = w_force(i,4);

    if w_force(i,1) ~= 0
        force_data(j1*3-2, 1) = w_force(i,1);
    end

    if w_force(i,2) ~= 0
        force_data(j1*3-1, 1) = w_force(i,2);
    end

    if w_force(i,3) ~= 0
        force_data(j1*3-0, 1) = w_force(i,3);
    end

end

%get joints with constraints

%move through the matrix backwards.
%this way the numbering scheme is easier to control
[a b] = size(w_constraint);

[ad_size b] = size(all_data);
[f_length b2] = size(force_data);

%how big does the matrix need to be?
mod_size = 0;

for i = 1:a

    j1 = w_constraint(i,4);

```

```

% x constraint
if w_constraint(i,1) == 1

    mod_size = mod_size+1;

    all_data(j1*3-2,b+mod_size) = 1;
end

% y constraint
if w_constraint(i,2) == 1

    mod_size = mod_size+1;

    all_data(j1*3-1,b+mod_size) = 1;
end

% z constraint
if w_constraint(i,3) == 1

    mod_size = mod_size+1;

    all_data(j1*3-0,b+mod_size) = 1;
end
end

% %condition data for 3d / 2d
if get(handles.user_2d, 'Value') == 1
    %get 2d plane

    if get(handles.user_2dxy, 'Value') == 1
        offset = 0;
    elseif get(handles.user_2dxz, 'Value') == 1
        offset = 1;
    elseif get(handles.user_2dyz, 'Value') == 1
        offset = 2;
    end

    %go backwards to maintain numbering system
    for i = j_length*3:-3:3
        all_data(i-offset,:) = [];
        force_data(i-offset,:) = [];
    end
end

reaction_force = all_data\(-force_data);

%set matrix to world
w_all_data = all_data;
w_reaction_force = reaction_force;
w_force_data = force_data;

```

```

%draw value
%
[a b] = size(all_data);

%get display info to draw

switch get(handles.ShowJointMenu,'Checked')
    case 'on'
        tog_jointnum = 1;
    case 'off'
        tog_jointnum = 0;
    end

switch get(handles.ShowMemberMenu,'Checked')
    case 'on'
        tog_membernum = 1;
    case 'off'
        tog_membernum = 0;
    end

switch get(handles.ShowConstraintMenu,'Checked')
    case 'on'
        tog_constraint = 1;
    case 'off'
        tog_constraint = 0;
    end

switch get(handles.ShowForceMenu,'Checked')
    case 'on'
        tog_force = 1;
    case 'off'
        tog_force = 0;
    end

switch get(handles.ShowGridMenu,'Checked')
    case 'on'
        tog_grid = 1;
    case 'off'
        tog_grid = 0;
    end

switch get(handles.ShowAxisLabelMenu,'Checked')
    case 'on'
        tog_axis_label = 1;
    case 'off'
        tog_axis_label = 0;
    end

switch get(handles.ShowAxesMenu,'Checked')
    case 'on'
        tog_axes = 1;
    case 'off'

```

```

    tog_axes = 0;
end

%reset view, no
tog_view = 1;

axes(handles.axes1);
user_axis = draw_plot( tog_jointnum, tog_membernum, tog_constraint, tog_force, tog_grid,
tog_axis_label, tog_axes, tog_view,all_data,reaction_force);

rotate3d on

axis(user_axis)

%use all_data to draw FBDs of each Joint
if (get(handles.check_drawFBD,'Value')) == 1

    %loop through all joints, draw all members attached to the joint.
    % draw members
    [a b] = size(w_member);
    [j_num b] = size(w_joint);

    %search all joints
    for kk = 1:j_num

        figure(kk);
        clf reset;
        hold on;

        %draw joint number

        text(w_joint(kk,1),w_joint(kk,2),w_joint(kk,3), num2str(kk),...
            'BackgroundColor',[.1 .5 .2],...
            'VerticalAlignment','bottom')

    for i = 1:a

        j1_temp = w_member(i,1);
        j2_temp = w_member(i,2);

        match = 0;

        if j1_temp == kk
            j1 = w_member(i,1);
            j2 = w_member(i,2);

            match = 1;
        elseif j2_temp == kk
            j2 = w_member(i,1);
            j1 = w_member(i,2);

            match = 1;
        end
    end
end

```

```

end

if match == 1
    xm = [ w_joint(j1,1) w_joint(j2,1) ];
    ym = [ w_joint(j1,2) w_joint(j2,2) ];
    zm = [ w_joint(j1,3) w_joint(j2,3) ];

    plot3(xm,ym,zm,'b-','LineWidth',5)
    plot3(w_joint(j2,1), w_joint(j2,2),
w_joint(j2,3),'bd','MarkerEdgeColor','k','MarkerFaceColor','b','MarkerSize',10)

    %draw member number
    if tog_membernum == 1
        text(mean(xm),mean(ym),mean(zm),num2str(i),...
            'BackgroundColor',[.7 .3 .2],...
            'VerticalAlignment','bottom')
    end
end
end

end

max_size = 1.5;
scale_force = max([max(max(abs(w_force))) max(abs(reaction_force)) ]) ;

%draw forces
[a b] = size(w_force);
for jj = 1:a

    index = w_force(jj,4);

    figure(index)
    hold on;

    %get axis info to scale force vector and constraints
    user_axis = axis;

    x_length = user_axis(2) - user_axis(1);
    y_length = user_axis(4) - user_axis(3);

    if length(user_axis) == 6
        z_length = user_axis(6) - user_axis(5);
    else
        z_length = 0;
    end

    %scale graphics
    scale_line = sqrt(x_length^2 + y_length^2 + z_length^2);

    xf_scale = max_size*scale_line/scale_force;

```



```

yf_scale = max_size*scale_line/scale_force;
zf_scale = max_size*scale_line/scale_force;

xf(1) = w_joint(index,1);
yf(1) = w_joint(index,2);
zf(1) = w_joint(index,3);

xf(2) = xf(1) + w_force(jj,1)*xf_scale;
yf(2) = yf(1) + w_force(jj,2)*yf_scale;
zf(2) = zf(1) + w_force(jj,3)*zf_scale;

%draw FX
plot3([xf(1) xf(2)], [yf(1) yf(1)], [zf(1) zf(1)], 'r-', 'LineWidth', 1)

%draw FY
plot3([xf(1) xf(1)], [yf(1) yf(2)], [zf(1) zf(1)], 'r-', 'LineWidth', 1)

%draw FZ
plot3([xf(1) xf(1)], [yf(1) yf(1)], [zf(1) zf(2)], 'r-', 'LineWidth', 1)

%draw diagonal
plot3(xf, yf, zf, 'r-', 'LineWidth', 3)
plot3( xf(2), yf(2), zf(2), 'rd', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'r', 'MarkerSize', 10)

end

[m_length b] = size(w_member);
[rf_length b] = size(reaction_force);

count = m_length+1;

%draw constraints
[a b] = size(w_constraint);

for i = 1:a

    index = w_constraint(i,4);

    figure(index)
    hold on;

    %get axis info to scale force vector and constraints
    user_axis = axis;

    x_length = user_axis(2) - user_axis(1);
    y_length = user_axis(4) - user_axis(3);

    if length(user_axis) == 6

```

```

    z_length = user_axis(6) - user_axis(5);
else
    z_length = 0;
end

xf_scale = max_size*scale_line/scale_force;
yf_scale = max_size*scale_line/scale_force;
zf_scale = max_size*scale_line/scale_force;

%scale graphics
scale_line = sqrt(x_length^2 + y_length^2 + z_length^2);

xf(1) = w_joint(index,1);
yf(1) = w_joint(index,2);
zf(1) = w_joint(index,3);

xf(2) = xf(1);
yf(2) = yf(1);
zf(2) = zf(1);

if w_constraint(i,1) == 1
    xf(2) = xf(1) + reaction_force(count)*xf_scale;

    %draw FX

    plot3(xf, yf, zf,'g-','LineWidth',3)
    plot3( xf(2), yf(2), zf(2),...
        'gd','MarkerEdgeColor',...
        'k','MarkerFaceColor','g',...
        'MarkerSize',10)

    val = num2str(reaction_force(count),'%7.1f');

    count = count + 1;

end

xf(2) = xf(1);

if w_constraint(i,2) == 1
    yf(2) = yf(1) + reaction_force(count)*yf_scale;

    plot3(xf, yf, zf,'g-','LineWidth',3)
    plot3( xf(2), yf(2), zf(2),...
        'gd','MarkerEdgeColor',...

```

```

        'k','MarkerFaceColor','g',...
        'MarkerSize',10)

    val = num2str(reaction_force(count), '%7.1f');

    count = count + 1;
end

yf(2) = yf(1);

if w_constraint(i,3) == 1
    zf(2) = zf(1) + reaction_force(count)*zf_scale;

    %draw FZ
    plot3(xf, yf, zf,'g-','LineWidth',3)
    plot3( xf(2), yf(2), zf(2),...
        'gd','MarkerEdgeColor',...
        'k','MarkerFaceColor','g',...
        'MarkerSize',10)

    val = num2str(reaction_force(count), '%7.1f');

    count = count + 1;
end

end

for gg = 1:j_num

    figure(gg)
    hold on;

    axis tight

    axis equal
    grid on;
    xlabel('X axis')
    ylabel('Y axis')
    zlabel('Z axis')

    view(3);

    temp_axis = axis;

    if temp_axis(1) > 0

```

```

        temp_axis(1) = temp_axis(1) * 0.85;
    else
        temp_axis(1) = temp_axis(1) * 1.15;
    end

    if temp_axis(2) < 0
        temp_axis(2) = temp_axis(2) * 0.85;
    else
        temp_axis(2) = temp_axis(2) * 1.15;
    end

    if temp_axis(3) > 0
        temp_axis(3) = temp_axis(3) * 0.85;
    else
        temp_axis(3) = temp_axis(3) * 1.15;
    end

    if temp_axis(4) < 0
        temp_axis(4) = temp_axis(4) * 0.85;
    else
        temp_axis(4) = temp_axis(4) * 1.15;
    end

    if length(temp_axis) == 86
        if temp_axis(5) > 0
            temp_axis(5) = temp_axis(5) * 0.85;
        else
            temp_axis(5) = temp_axis(5) * 1.15;
        end

        if temp_axis(6) < 0
            temp_axis(6) = temp_axis(6) * 0.85;
        else
            temp_axis(6) = temp_axis(6) * 1.15;
        end
    end

    temp_axis = temp_axis * 1.15;
    axis(temp_axis)
    rotate3d on;

end

end

% axes(handles.axes1);
% cla;

```

```

%
% popup_sel_index = get(handles.listbox3, 'Value');
% switch popup_sel_index
%   case 1
%       plot(sin(1:0.01:25));
%   case 2
%       comet(cos(1:0.01:10));
%   case 3
%       plot(membrane);
%   case 4
%       surf(peaks);
%   case 5
%       plot(sin(1:0.01:25));
%       text(550,0,'text')
%
%
%
% end
%
% beep

% -----
function FileMenu_Callback(hObject, eventdata, handles)
% hObject    handle to FileMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% -----
function OpenMenuItem_Callback(hObject, eventdata, handles)
% hObject    handle to OpenMenuItem (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global w_joint w_member w_constraint w_force

[a b] = size(w_joint);

if a ~= 0

    button = questdlg('Do you want to save the model?',...
        'Save Model?', 'Yes', 'No', 'No');
    if strcmp(button, 'Yes')

        [file, path] = uiputfile({'*.mat', 'Matlab Model File (*.mat)'}, 'Save As');

        if ~isequal(file, 0)

            filename = char(file);
            %load file
            save(filename);

```

```

        %redraw all
        %axes(handles.axes1);
        new_axis = draw_plot( 1, 1, 1, 1, 1, 1, 1, 1,1,1);
        rotate3d on
    end
elseif strcmp(button,'No')
    %

end

end

file = uigetfile({'*.mat','Matlab Model File (*.mat)'},'Open Model');

if ~isequal(file, 0)
    %open(file)

    %load file
    load(char(file));

    %redraw all
    %axes(handles.axes1);
    new_axis = draw_plot( 1, 1, 1, 1, 1, 1, 1, 1,1,1);
    new_axis = draw_plot( 1, 1, 1, 1, 1, 1, 1, 1,1,1);
    new_axis = draw_plot( 1, 1, 1, 1, 1, 1, 1, 1,1,1);

    rotate3d on

end

% -----
function PrintMenuItem_Callback(hObject, eventdata, handles)
% hObject    handle to PrintMenuItem (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
printdlg(handles.figure1)

% -----
function CloseMenuItem_Callback(hObject, eventdata, handles)
% hObject    handle to CloseMenuItem (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
selection = questdlg(['Close ' get(handles.figure1,'Name') '?'],...
    ['Close ' get(handles.figure1,'Name') '...'],...
    'Yes','No','Yes');
if strcmp(selection,'No')
    return;
end

```

```

delete(handles.figure1)

clear all

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in user_details.
function user_details_Callback(hObject, eventdata, handles)
% hObject    handle to user_details (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global w_joint w_member w_constraint w_force w_all_data w_reaction_force
is_error = 0;

%output model information to Command Window

prompt = {'Model / Project Name:', 'Your Name:', 'Team:', 'Description', 'Units:'};
dlg_title = 'Model Details';
num_lines = 1;
def = {'Bridge Team Project', 'not Rooney', 'The Nuclear Orange Cheeto Fingers', 'Truss Analysis of Team Project', 'Force (Newtons)          Length (feet)'};
answer = inputdlg(prompt, dlg_title, num_lines, def);

%only if input exists
if length(answer) > 0

    fid = fopen('EFD_TRUSS_output.txt', 'wb');

    clc

```

```

fprintf(fid,'\n-----');

fprintf(fid,'\n-----');
fprintf(fid,'\n-----');

fprintf(fid,'\n\n-- Model / Project Name : \n\n      ');
fprintf(fid,(char(answer(1))));
fprintf(fid,'\n\n-- Name :          \n\n      ');
%disp(char(answer(2)))
fprintf(fid,(char(answer(2))));
fprintf(fid,'\n\n-- Team :          \n\n      ');
fprintf(fid,(char(answer(3))));
fprintf(fid,'\n\n-- Description :      \n\n      ');
fprintf(fid,(char(answer(4))));
fprintf(fid,'\n\n-- Units:          \n\n      ');
fprintf(fid,(char(answer(5))));

fprintf(fid,'\n\n-----');
fprintf(fid,'\n-----JOINTS-----');
fprintf(fid,'\n-----');

fprintf(fid,'\n\n      num      X-Pos      Y-Pos      Z-Pos');
fprintf(fid,'\n-----\n');

[a b] = size(w_joint);

for i = 1:a
    fprintf(fid,'\n      %3.0f      %7.2f      %7.2f      %7.2f,i,w_joint(i,:));
end

fprintf(fid,'\n\n-----');
fprintf(fid,'\n-----MEMBERS-----');
fprintf(fid,'\n-----');

fprintf(fid,'\n\n      num      Joint 1      Joint 2      ');
fprintf(fid,'\n-----\n');

[a b] = size(w_member);

% remove blank user_z value
temp_member(:,1) = w_member(:,1);
temp_member(:,2) = w_member(:,2);

for i = 1:a
    fprintf(fid,'\n      %3.0f      %3.0f      %3.0f      ',i,temp_member(i,:));
end

fprintf(fid,'\n\n-----');
fprintf(fid,'\n-----CONSTRAINTS-----');
fprintf(fid,'\n-----');

```



```

fprintf(fid,'\n\n      num      CX      CY      CZ      Joint Num');
fprintf(fid,'\n-----\n');

[a b] = size(w_constraint);

for i = 1:a
    fprintf(fid,'\n      %3.0f      %3.0f %3.0f %3.0f      %3.0f', i, w_constraint(i,:));
end

fprintf(fid,'\n\n-----');
fprintf(fid,'\n-----FORCES-----');
fprintf(fid,'\n-----');

fprintf(fid,'\n\n num      FX      FY      FZ      Joint Num');
fprintf(fid,'\n-----\n');

[a b] = size(w_force);

for i = 1:a
    fprintf(fid,'\n %3.0f      %9.2f %9.2f %9.2f      %3.0f', i, w_force(i,:));
end

fprintf(fid,'\n\n-----');
fprintf(fid,'\n-----');
fprintf(fid,'\n-----\n');

%write analysis if it exists

if isnan(w_all_data) ~= 1
    fprintf(fid,'\n\n-----');
    fprintf(fid,'\n-----ANALYSIS-----');
    fprintf(fid,'\n-----');

    fprintf(fid,'\n\n      Reaction Forces');
    fprintf(fid,'\n-----\n');

    [m_length b] = size(w_member);
    [rf_length b] = size(w_reaction_force);

    % Write member loads

    for k = 1:m_length

        fprintf(fid,'\n      %9.2f', abs(w_reaction_force(k)));

        if w_reaction_force(k) > 0
            fprintf(fid,' T ', w_reaction_force(k));
        end
    end

```

```

elseif w_reaction_force(k) == 0
    fprintf(fid,' - ', w_reaction_force(k));
else
    fprintf(fid,' C ', w_reaction_force(k));
end

fprintf(fid,' Member # %3.0f', k);

end

count = m_length+1;

%write constraint reactions
[a b] = size(w_constraint);

for i = 1:a

    index = w_constraint(i,4);

    if w_constraint(i,1) == 1

        fprintf(fid,'\n      %9.2f', (w_reaction_force(count)));

        %      if w_reaction_force(count) > 0
        %          fprintf(' T ', w_reaction_force(count));
        %      elseif w_reaction_force(count) == 0
        %          fprintf(' - ', w_reaction_force(count));
        %      else
        %          fprintf(' C ', w_reaction_force(count));
        %      end

        fprintf(fid,'    X constraint at Joint %3.0f', index);

        count = count + 1;

    end

    if w_constraint(i,2) == 1
        fprintf(fid,'\n      %9.2f', (w_reaction_force(count)));

        %      if w_reaction_force(count) > 0
        %          fprintf(' T ', w_reaction_force(count));
        %      elseif w_reaction_force(count) == 0

```

```

%         fprintf(' - ', w_reaction_force(count));
%     else
%         fprintf(' C ', w_reaction_force(count));
%     end

fprintf(fid,'    Y constraint at Joint %3.0f, index);

count = count + 1;

end

if w_constraint(i,3) == 1
    fprintf(fid,'\n    %9.2f', (w_reaction_force(count)));

    %     if w_reaction_force(count) > 0
    %         fprintf(' T ', w_reaction_force(count));
    %     elseif w_reaction_force(count) == 0
    %         fprintf(' - ', w_reaction_force(count));
    %     else
    %         fprintf(' C ', w_reaction_force(count));
    %     end

    fprintf(fid,'    Z constraint at Joint %3.0f, index);

    count = count + 1;
end
end

end

fprintf(fid,'\n\n-----');
fprintf(fid,'\n-----');
fprintf(fid,'\n-----\n');

% [a b] = size(w_reaction_force);
%
% for i = 1:a
%     fprintf('\n')
%     for j = 1:b
%         fprintf('    %9.2f, i, w_all_data(i,j))
%     end
% end

%close file
fclose('all');
```

```

    %print text file to HTML
    web(['file:' which('EFD_TRUSS_output.txt')],'-browser')

end

% -----
function ViewMenu_Callback(hObject, eventdata, handles)
% hObject   handle to ViewMenu (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

% -----
function DisplayMenu_Callback(hObject, eventdata, handles)
% hObject   handle to DisplayMenu (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

% -----
function ShowJointMenu_Callback(hObject, eventdata, handles)
% hObject   handle to ShowJointMenu (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

switch get(handles.ShowJointMenu,'Checked')
case 'on'
    set(handles.ShowJointMenu, 'Checked', 'off')
case 'off'
    set(handles.ShowJointMenu, 'Checked', 'on')
end

switch get(handles.ShowJointMenu,'Checked')
case 'on'
    tog_jointnum = 1;
case 'off'
    tog_jointnum = 0;
end

switch get(handles.ShowMemberMenu,'Checked')
case 'on'
    tog_membernum = 1;
case 'off'

```

```

    tog_membernum = 0;
end

switch get(handles.ShowConstraintMenu,'Checked')
case 'on'
    tog_constraint = 1;
case 'off'
    tog_constraint = 0;
end

switch get(handles.ShowForceMenu,'Checked')
case 'on'
    tog_force = 1;
case 'off'
    tog_force = 0;
end

switch get(handles.ShowGridMenu,'Checked')
case 'on'
    tog_grid = 1;
case 'off'
    tog_grid = 0;
end

switch get(handles.ShowAxisLabelMenu,'Checked')
case 'on'
    tog_axis_label = 1;
case 'off'
    tog_axis_label = 0;
end

switch get(handles.ShowAxesMenu,'Checked')
case 'on'
    tog_axes = 1;
case 'off'
    tog_axes = 0;
end

%reset view
tog_view = 0;

global w_all_data w_reaction_force

%check for analysis, draw
if sum(w_all_data) == 0

    all_data = 1;
    reaction_force = 1;
else

    all_data = w_all_data;
    reaction_force = w_reaction_force;
end

```

```

end

axes(handles.axes1);
new_axis = draw_plot( tog_jointnum, tog_membernum, tog_constraint, tog_force, tog_grid,
tog_axis_label, tog_axes, tog_view, all_data, reaction_force);
rotate3d on

% -----
function ShowMemberMenu_Callback(hObject, eventdata, handles)
% hObject   handle to ShowMemberMenu (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

switch get(handles.ShowMemberMenu,'Checked')
case 'on'
    set(handles.ShowMemberMenu, 'Checked', 'off')
case 'off'
    set(handles.ShowMemberMenu, 'Checked', 'on')
end

switch get(handles.ShowJointMenu,'Checked')
case 'on'
    tog_jointnum = 1;
case 'off'
    tog_jointnum = 0;
end

switch get(handles.ShowMemberMenu,'Checked')
case 'on'
    tog_membernum = 1;
case 'off'
    tog_membernum = 0;
end

switch get(handles.ShowConstraintMenu,'Checked')
case 'on'
    tog_constraint = 1;
case 'off'
    tog_constraint = 0;
end

switch get(handles.ShowForceMenu,'Checked')
case 'on'
    tog_force = 1;
case 'off'
    tog_force = 0;
end

switch get(handles.ShowGridMenu,'Checked')
case 'on'
    tog_grid = 1;
case 'off'
    tog_grid = 0;
end

```

```

end

switch get(handles.ShowAxisLabelMenu,'Checked')
case 'on'
    tog_axis_label = 1;
case 'off'
    tog_axis_label = 0;
end

switch get(handles.ShowAxesMenu,'Checked')
case 'on'
    tog_axes = 1;
case 'off'
    tog_axes = 0;
end

%reset view
tog_view = 0;

global w_all_data w_reaction_force

%check for analysis, draw
if sum(w_all_data) == 0

    all_data = 1;
    reaction_force = 1;
else

    all_data = w_all_data;
    reaction_force = w_reaction_force;

end

axes(handles.axes1);
new_axis = draw_plot( tog_jointnum, tog_membernum, tog_constraint, tog_force, tog_grid,
tog_axis_label, tog_axes, tog_view, all_data, reaction_force);
rotate3d on

% -----
function ShowConstraintMenu_Callback(hObject, eventdata, handles)
% hObject    handle to ShowConstraintMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

switch get(handles.ShowConstraintMenu,'Checked')
case 'on'
    set(handles.ShowConstraintMenu, 'Checked', 'off')
case 'off'
    set(handles.ShowConstraintMenu, 'Checked', 'on')
end

switch get(handles.ShowJointMenu,'Checked')
case 'on'

```

```

    tog_jointnum = 1;
case 'off'
    tog_jointnum = 0;
end

switch get(handles.ShowMemberMenu,'Checked')
case 'on'
    tog_membernum = 1;
case 'off'
    tog_membernum = 0;
end

switch get(handles.ShowConstraintMenu,'Checked')
case 'on'
    tog_constraint = 1;
case 'off'
    tog_constraint = 0;
end

switch get(handles.ShowForceMenu,'Checked')
case 'on'
    tog_force = 1;
case 'off'
    tog_force = 0;
end

switch get(handles.ShowGridMenu,'Checked')
case 'on'
    tog_grid = 1;
case 'off'
    tog_grid = 0;
end

switch get(handles.ShowAxisLabelMenu,'Checked')
case 'on'
    tog_axis_label = 1;
case 'off'
    tog_axis_label = 0;
end

switch get(handles.ShowAxesMenu,'Checked')
case 'on'
    tog_axes = 1;
case 'off'
    tog_axes = 0;
end

%reset view
tog_view = 0;

global w_all_data w_reaction_force

%check for analysis, draw

```



```

if sum(w_all_data) == 0

    all_data = 1;
    reaction_force = 1;
else

    all_data = w_all_data;
    reaction_force = w_reaction_force;

end

axes(handles.axes1);
new_axis = draw_plot( tog_jointnum, tog_membernum, tog_constraint, tog_force, tog_grid,
tog_axis_label, tog_axes, tog_view, all_data, reaction_force);
rotate3d on

% -----
function ShowForceMenu_Callback(hObject, eventdata, handles)
% hObject    handle to ShowForceMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

switch get(handles.ShowForceMenu,'Checked')
case 'on'
    set(handles.ShowForceMenu, 'Checked', 'off')
case 'off'
    set(handles.ShowForceMenu, 'Checked', 'on')
end

switch get(handles.ShowJointMenu,'Checked')
case 'on'
    tog_jointnum = 1;
case 'off'
    tog_jointnum = 0;
end

switch get(handles.ShowMemberMenu,'Checked')
case 'on'
    tog_membernum = 1;
case 'off'
    tog_membernum = 0;
end

switch get(handles.ShowConstraintMenu,'Checked')
case 'on'
    tog_constraint = 1;
case 'off'
    tog_constraint = 0;
end

switch get(handles.ShowForceMenu,'Checked')
case 'on'

```

```

        tog_force = 1;
    case 'off'
        tog_force = 0;
    end

    switch get(handles.ShowGridMenu,'Checked')
    case 'on'
        tog_grid = 1;
    case 'off'
        tog_grid = 0;
    end

    switch get(handles.ShowAxisLabelMenu,'Checked')
    case 'on'
        tog_axis_label = 1;
    case 'off'
        tog_axis_label = 0;
    end

    switch get(handles.ShowAxesMenu,'Checked')
    case 'on'
        tog_axes = 1;
    case 'off'
        tog_axes = 0;
    end

    %reset view
    tog_view = 0;

    global w_all_data w_reaction_force

    %check for analysis, draw
    if sum(w_all_data) == 0

        all_data = 1;
        reaction_force = 1;
    else

        all_data = w_all_data;
        reaction_force = w_reaction_force;
    end

    axes(handles.axes1);
    new_axis = draw_plot( tog_jointnum, tog_membernum, tog_constraint, tog_force, tog_grid,
    tog_axis_label, tog_axes, tog_view, all_data, reaction_force);
    rotate3d on

    % -----
    function ShowGridMenu_Callback(hObject, eventdata, handles)
    % hObject    handle to ShowGridMenu (see GCBO)
    % eventdata reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)

```

```

switch get(handles.ShowGridMenu,'Checked')
case 'on'
    set(handles.ShowGridMenu, 'Checked', 'off')
case 'off'
    set(handles.ShowGridMenu, 'Checked', 'on')
end

switch get(handles.ShowJointMenu,'Checked')
case 'on'
    tog_jointnum = 1;
case 'off'
    tog_jointnum = 0;
end

switch get(handles.ShowMemberMenu,'Checked')
case 'on'
    tog_membernum = 1;
case 'off'
    tog_membernum = 0;
end

switch get(handles.ShowConstraintMenu,'Checked')
case 'on'
    tog_constraint = 1;
case 'off'
    tog_constraint = 0;
end

switch get(handles.ShowForceMenu,'Checked')
case 'on'
    tog_force = 1;
case 'off'
    tog_force = 0;
end

switch get(handles.ShowGridMenu,'Checked')
case 'on'
    tog_grid = 1;
case 'off'
    tog_grid = 0;
end

switch get(handles.ShowAxisLabelMenu,'Checked')
case 'on'
    tog_axis_label = 1;
case 'off'
    tog_axis_label = 0;
end

switch get(handles.ShowAxesMenu,'Checked')
case 'on'
    tog_axes = 1;

```

```

case 'off'
    tog_axes = 0;
end

%reset view
tog_view = 0;

global w_all_data w_reaction_force

%check for analysis, draw
if sum(w_all_data) == 0

    all_data = 1;
    reaction_force = 1;
else

    all_data = w_all_data;
    reaction_force = w_reaction_force;

end

axes(handles.axes1);
new_axis = draw_plot( tog_jointnum, tog_membernum, tog_constraint, tog_force, tog_grid,
tog_axis_label, tog_axes, tog_view, all_data, reaction_force);
rotate3d on

% -----
function ErrorCheckMenu_Callback(hObject, eventdata, handles)
% hObject    handle to ErrorCheckMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% -----
function CheckJointMenu_Callback(hObject, eventdata, handles)
% hObject    handle to CheckJointMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% -----
function CheckMemberMenu_Callback(hObject, eventdata, handles)
% hObject    handle to CheckMemberMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% -----
function CheckConstraintsMenu_Callback(hObject, eventdata, handles)
% hObject    handle to CheckConstraintsMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

% -----
function HelpMenu_Callback(hObject, eventdata, handles)
% hObject   handle to HelpMenu (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

% -----
function HelpGuideMenu_Callback(hObject, eventdata, handles)
% hObject   handle to HelpGuideMenu (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

web(['file:' which('help_home.html')],'-browser')
%web('help_home.htm','-browser')

% -----
function HelpAboutMenu_Callback(hObject, eventdata, handles)
% hObject   handle to HelpAboutMenu (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

msgbox('Method of Joints Truss Solver,                               written by Jon Huber', 'About this
program','help')

% --- Executes during object creation, after setting all properties.
function listBox3_CreateFcn(hObject, eventdata, handles)
% hObject   handle to listBox3 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   empty - handles not created until after all CreateFns called

% Hint: listBox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

set(hObject, 'String', {' Joint', ' Member', ' Constraint', ' Force'});

% --- Executes on selection change in listBox3.
function listBox3_Callback(hObject, eventdata, handles)
% hObject   handle to listBox3 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns listBox3 contents as cell array
%       contents{get(hObject,'Value')} returns selected item from listBox3

```

```

popup_sel_index = get(handles.listbox3, 'Value');
switch popup_sel_index
case 1
    %joint
    set(handles.text13, 'String', 'X pos');
    set(handles.text14, 'String', 'Y pos');
    set(handles.text15, 'String', 'Z pos');
    set(handles.user_x, 'String', "");
    set(handles.user_y, 'String', "");
    set(handles.user_z, 'String', "");
    %set(handles.mod_num, 'String', "");

    set(handles.text28, 'String', 'Feature Number');
case 2
    %member
    set(handles.text13, 'String', 'Joint 1');
    set(handles.text14, 'String', 'Joint 2');
    set(handles.text15, 'String', '');
    set(handles.user_x, 'String', "");
    set(handles.user_y, 'String', "");
    set(handles.user_z, 'String', 'XXXXXXXXX');
    set(handles.mod_num, 'String', "");

    set(handles.text28, 'String', 'Feature Number');

case 3
    %constraint
    set(handles.text13, 'String', 'Con X');
    set(handles.text14, 'String', 'Con Y');
    set(handles.text15, 'String', 'Con Z');
    set(handles.user_x, 'String', "");
    set(handles.user_y, 'String', "");
    set(handles.user_z, 'String', "");
    %set(handles.mod_num, 'String', "");

    set(handles.text28, 'String', 'Joint Number');
case 4
    %force
    set(handles.text13, 'String', 'F X');
    set(handles.text14, 'String', 'F Y');
    set(handles.text15, 'String', 'F Z');
    set(handles.user_x, 'String', "");
    set(handles.user_y, 'String', "");
    set(handles.user_z, 'String', "");
    %set(handles.mod_num, 'String', "");

    set(handles.text28, 'String', 'Joint Number');
end

```

% --- Executes during object creation, after setting all properties.

```

function edit12_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

```

```

function edit12_Callback(hObject, eventdata, handles)
% hObject    handle to edit12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit12 as text
%       str2double(get(hObject,'String')) returns contents of edit12 as a double

```

```

% --- Executes during object creation, after setting all properties.
function edit14_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

```

```

function edit14_Callback(hObject, eventdata, handles)
% hObject    handle to edit14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit14 as text
%       str2double(get(hObject,'String')) returns contents of edit14 as a double

```

```

% --- Executes on button press in radiobutton1.
function radiobutton1_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB

```

```

% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton1

% --- Executes on button press in radiobutton2.
function radiobutton2_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton2

% --- Executes during object creation, after setting all properties.
function edit19_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit19 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit19_Callback(hObject, eventdata, handles)
% hObject    handle to edit19 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit19 as text
%        str2double(get(hObject,'String')) returns contents of edit19 as a double

% --- Executes during object creation, after setting all properties.
function edit20_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit20 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

```



```

function edit20_Callback(hObject, eventdata, handles)
% hObject    handle to edit20 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit20 as text
%        str2double(get(hObject,'String')) returns contents of edit20 as a double

% -----
function ViewFrontMenu_Callback(hObject, eventdata, handles)
% hObject    handle to ViewFrontMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

az = 0;
el = 0;
view(az, el);

% -----
function ViewTopMenu_Callback(hObject, eventdata, handles)
% hObject    handle to ViewTopMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

az = 0;
el = 90;
view(az, el);

% -----
function ViewSideMenu_Callback(hObject, eventdata, handles)
% hObject    handle to ViewSideMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

az = 90;
el = 0;
view(az, el);

% -----
function ViewIsoMenu_Callback(hObject, eventdata, handles)
% hObject    handle to ViewIsoMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

az = 30;
el = 30;
view(az, el);

% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton7 (see GCBO)

```

```

% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function user_x_CreateFcn(hObject, eventdata, handles)
% hObject handle to user_x (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function user_x_Callback(hObject, eventdata, handles)
% hObject handle to user_x (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of user_x as text
% str2double(get(hObject,'String')) returns contents of user_x as a double

% --- Executes during object creation, after setting all properties.
function user_y_CreateFcn(hObject, eventdata, handles)
% hObject handle to user_y (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function user_y_Callback(hObject, eventdata, handles)
% hObject handle to user_y (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of user_y as text
% str2double(get(hObject,'String')) returns contents of user_y as a double

```

```
% --- Executes during object creation, after setting all properties.
function user_z_CreateFcn(hObject, eventdata, handles)
% hObject    handle to user_z (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function user_z_Callback(hObject, eventdata, handles)
% hObject    handle to user_z (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of user_z as text
%       str2double(get(hObject,'String')) returns contents of user_z as a double
```

```
% --- Executes during object creation, after setting all properties.
function mod_num_CreateFcn(hObject, eventdata, handles)
% hObject    handle to mod_num (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function mod_num_Callback(hObject, eventdata, handles)
% hObject    handle to mod_num (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of mod_num as text
%       str2double(get(hObject,'String')) returns contents of mod_num as a double
```

```
% --- Executes on button press in mod_add.
function mod_add_Callback(hObject, eventdata, handles)
```

```

% hObject   handle to mod_add (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of mod_add
set(handles.mod_add, 'Value', 1);
set(handles.mod_mod, 'Value', 0);
set(handles.mod_remove, 'Value', 0);

% --- Executes on button press in mod_mod.
function mod_mod_Callback(hObject, eventdata, handles)
% hObject   handle to mod_mod (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of mod_mod
set(handles.mod_add, 'Value', 0);
set(handles.mod_mod, 'Value', 1);
set(handles.mod_remove, 'Value', 0);

% --- Executes on button press in mod_remove.
function mod_remove_Callback(hObject, eventdata, handles)
% hObject   handle to mod_remove (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of mod_remove
set(handles.mod_add, 'Value', 0);
set(handles.mod_mod, 'Value', 0);
set(handles.mod_remove, 'Value', 1);

% --- Executes on button press in user_apply.
function user_apply_Callback(hObject, eventdata, handles)
% hObject   handle to user_apply (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

global w_joint w_member w_constraint w_force w_mod_num

is_error = 0;

%
%
% search for specifed feature
%
%
% is it Add / Modify / Remove ???
%

% check for legal feature entry and add

w_mod_num = str2double(get(handles.mod_num,'String'));

```

```

% get values
user_x = str2double(get(handles.user_x,'String'));
user_y = str2double(get(handles.user_y,'String'));
user_z = str2double(get(handles.user_z,'String'));

% if member
if get(handles.listbox3,'Value') == 2
    user_z = 0;
end

% check for NaN
%if mod or remove, w_mod_num must be legal
if (get(handles.mod_remove,'Value')) == 1
    if isnan(w_mod_num) == 1
        errordlg('You have entered an illegal Feature / Joint value','User Input Error');
        is_error = 1;
        return;
    end
elseif isnan(user_x) == 1 | isnan(user_y) == 1 | isnan(user_z) == 1
    errordlg('You have entered an illegal feature value','User Input Error');
    is_error = 1;
    return;
end

if w_mod_num == 0
    errordlg('The feature value will never exist','User Input Error');
    is_error = 1;
    return;
end

%
%
%check feature type
popup_sel_index = get(handles.listbox3, 'Value');
switch popup_sel_index
case 1
    %joint
    [a b] = size(w_joint);

    %check for pre-existing and add

    if (get(handles.mod_remove,'Value')) == 1 & a >= w_mod_num
        button = questdlg('Do you want to continue?',...
            'Remove Joint?','Yes','No','No');

        if strcmp(button,'Yes')
            %write updating function
            w_joint(w_mod_num,:) = [];

            %work through matrix, lower joint numbers above the deleted

```

```

    %remove the features connected to the joint
    tog_member = 1;
    w_member = subtract_joint(w_member, tog_member, w_mod_num);

    tog_member = 0;
    w_constraint = subtract_joint(w_constraint, tog_member, w_mod_num);
    w_force = subtract_joint(w_force, tog_member, w_mod_num);

    elseif strcmp(button,'No')
        % do nothing
        is_error = 1;
    end
elseif (get(handles.mod_remove,'Value')) == 1 & a < w_mod_num
    errorldg('The joint number does not exist','User Input Error');
    is_error = 1;
    return;
elseif (get(handles.mod_add,'Value')) == 1

    size_check1 = size(w_joint);
    w_joint = check_add(w_joint, user_x, user_y, user_z);
    size_check2 = size(w_joint);

    if size_check1 == size_check2
        is_error = 1;
        return;
    end
    %modify
elseif (get(handles.mod_mod,'Value')) == 1 & a >= w_mod_num
    %
    w_joint(w_mod_num,:) = [user_x user_y user_z];

elseif (get(handles.mod_mod,'Value')) == 1
    errorldg('The joint number does not exist','User Input Error');
    is_error = 1;
    return;
end

case 2
    %member
    [a b] = size(w_joint);
    [a2 b2] = size(w_member);

    %remove first
    if (get(handles.mod_remove,'Value')) == 1 & a2 >= w_mod_num
        button = questdlg('Do you want to continue?',...
            'Remove Member?','Yes','No','No');

        if strcmp(button,'Yes')

            w_member(w_mod_num,:) = [];

        elseif strcmp(button,'No')

```

```

        % do nothing
        is_error = 1;
        return;
    end
elseif (get(handles.mod_remove,'Value')) == 1 & a2 < w_mod_num
    errordlg('The member number does not exist','User Input Error');
    is_error = 1;
    return;
elseif isnan(user_x) == 1 | isnan(user_y) == 1
    errordlg('Enter a joint number','User Input Error');
    is_error = 1;
    return;
elseif user_x > a | user_y > a
    errordlg('The joint number does not exist','User Input Error');
    is_error = 1;
    return;
elseif user_x == user_y
    errordlg('The joint numbers cannot be equal','User Input Error');
    is_error = 1;
    return;
elseif user_x == 0 | user_y == 0
    errordlg('The joint number does not exist','User Input Error');
    is_error = 1;
    return;
elseif user_x ~= round(user_x) | user_y ~= round(user_y)
    errordlg('The joint number must be an integer','User Input Error');
    is_error = 1;
    return;

else
    %add
    if (get(handles.mod_add,'Value')) == 1
        % if sizes are equal, no addition occurred
        size_check1 = size(w_member);

        w_member = check_add(w_member, user_x, user_y, 0);

        size_check2 = size(w_member);

    %modify

    elseif (get(handles.mod_mod,'Value')) == 1 & a2 >= w_mod_num
        %
        w_member(w_mod_num,:) = [user_x user_y user_z];

    elseif (get(handles.mod_mod,'Value')) == 1
        errordlg('The member entry does not exist','User Input Error');
        is_error = 1;
        return;
    end
end
end

```

case 3

```
%constraint
[a b] = size(w_joint);
[a2 b2] = size(w_constraint);

check_constraint = 0;

%check that every value is 0 or 1
if (get(handles.mod_remove,'Value')) ~= 1
    if user_x == 1 | user_x == 0
        check_constraint = check_constraint + 1;
    end
    if user_y == 1 | user_y == 0
        check_constraint = check_constraint + 1;
    end
    if user_z == 1 | user_z == 0
        check_constraint = check_constraint + 1;
    end
end

%remove first
if (get(handles.mod_remove,'Value')) == 1 & a >= w_mod_num
    button = questdlg('Do you want to continue?',...
        'Remove Constraint?','Yes','No','No');

    if strcmp(button,'Yes')
        %search for existing point

        check_exist = 0;
        for i = 1:a2
            if w_constraint(i,4) == w_mod_num
                w_constraint(i,:) = [];
                check_exist = 1;
            end
        end

        break;
    end

    if check_exist == 0
        errordlg('The feature does not exist for this joint','User Input Error');
        is_error = 1;
        return;
    end

elseif strcmp(button,'No')
    % do nothing
    is_error = 1;
end
```



```

        return;
    end
elseif user_x + user_y + user_z == 0
    errordlg('Constraint values sum to 0, give a value or remove','User Input Error');
    is_error = 1;
elseif check_constraint == 3
    if isnan(w_mod_num) == 1
        errordlg('Enter a joint number','User Input Error');
        is_error = 1;
        return;
    end
end

if w_mod_num > a
    errordlg('The joint number does not exist','User Input Error');
    is_error = 1;
    return;
elseif isnan(w_mod_num) ~= 1

    %add
    if (get(handles.mod_add,'Value')) == 1
        size_check1 = size(w_constraint);
        w_constraint = check_add2(w_constraint, user_x, user_y, user_z, w_mod_num);
        size_check2 = size(w_constraint);

        if size_check1 == size_check2
            is_error = 1;
            return;
        end

        %modify
    elseif (get(handles.mod_mod,'Value')) == 1
        %
        %find joint, switch
        [a b] = size(w_constraint);
        check_exist = 0;

        for i = 1:a
            if w_constraint(i,4) == w_mod_num
                w_constraint(i,:) = [user_x user_y user_z w_mod_num];
                check_exist = 1;
            end
        end

        if check_exist == 0
            errordlg('The feature does not exist for this joint','User Input Error');
            is_error = 1;
            return;
        end
    end
end
end

```

```

else
    errordlg('Constraint must be a 0 or 1','User Input Error');
    is_error = 1;
    return;
end

case 4
    %force
    [a b] = size(w_joint);
    [a2 b2] = size(w_force);

    %remove first
    if (get(handles.mod_remove,'Value')) == 1 & a >= w_mod_num
        button = questdlg('Do you want to continue?',...
            'Remove Force?', 'Yes', 'No', 'No');

        if strcmp(button, 'Yes')

            %search for existing point
            check_exist = 0;
            for i = 1:a2
                if w_force(i,4) == w_mod_num

                    w_force(i,:) = [];
                    check_exist = 1;

                    break;
                end
            end

            if check_exist == 0
                errordlg('The feature does not exist for this joint','User Input Error');
                is_error = 1;
            end

            elseif strcmp(button, 'No')
                % do nothing
                is_error = 1;
            end
        elseif (get(handles.mod_remove,'Value')) == 1 & a < w_mod_num
            errordlg('The joint number does not exist','User Input Error');
            is_error = 1;
            return;
        elseif isnan(w_mod_num) == 1
            errordlg('Enter a joint number','User Input Error');
            is_error = 1;
            return;
        elseif w_mod_num > a
            errordlg('The joint number does not exist','User Input Error');

```

```

        is_error = 1;
        return;
    elseif isnan(w_mod_num) ~= 1
        %add
        if (get(handles.mod_add,'Value')) == 1
            size_check1 = size(w_force);
            w_force = check_add2(w_force, user_x, user_y, user_z, w_mod_num);
            size_check2 = size(w_force);

            if size_check1 == size_check2
                is_error = 1;
                return;
            end

            %modify
        elseif (get(handles.mod_mod,'Value')) == 1
            %
            %find joint, switch
            [a b] = size(w_force);
            check_exist = 0;

            for i = 1:a
                if w_force(i,4) == w_mod_num
                    w_force(i,:) = [user_x user_y user_z w_mod_num];
                    check_exist = 1;
                end
            end

            if check_exist == 0
                errordlg('The feature does not exist for this joint','User Input Error');
                is_error = 1;
                return;
            end

        end
    end
end

%if no error, redraw
if is_error == 0

    %get display info
    %tog_joint = (get(handles.ShowJointMenu,'Checked'))

    %send_axis = draw_plot( tog_jointnum, tog_membernum, tog_constraint, tog_force, tog_grid,
    tog_axis_label, tog_axes, tog_view)

    switch get(handles.ShowJointMenu,'Checked')
    case 'on'

```

```

        tog_jointnum = 1;
    case 'off'
        tog_jointnum = 0;
    end

    switch get(handles.ShowMemberMenu,'Checked')
    case 'on'
        tog_membernum = 1;
    case 'off'
        tog_membernum = 0;
    end

    switch get(handles.ShowConstraintMenu,'Checked')
    case 'on'
        tog_constraint = 1;
    case 'off'
        tog_constraint = 0;
    end

    switch get(handles.ShowForceMenu,'Checked')
    case 'on'
        tog_force = 1;
    case 'off'
        tog_force = 0;
    end

    switch get(handles.ShowGridMenu,'Checked')
    case 'on'
        tog_grid = 1;
    case 'off'
        tog_grid = 0;
    end

    switch get(handles.ShowAxisLabelMenu,'Checked')
    case 'on'
        tog_axis_label = 1;
    case 'off'
        tog_axis_label = 0;
    end

    switch get(handles.ShowAxesMenu,'Checked')
    case 'on'
        tog_axes = 1;
    case 'off'
        tog_axes = 0;
    end

    %reset view toggle
    tog_view = 1;

    axes(handles.axes1);

```

```

    new_axis = draw_plot( tog_jointnum, tog_membernum, tog_constraint, tog_force, tog_grid,
    tog_axis_label, tog_axes, tog_view,1,1);

```

```

    for i = 1:length(new_axis)
        new_axis(i) = round(new_axis(i)*10)/10;
    end

```

```

    set(handles.axis_xmin, 'String', new_axis(1));
    set(handles.axis_xmax, 'String', new_axis(2));
    set(handles.axis_ymin, 'String', new_axis(3));
    set(handles.axis_ymax, 'String', new_axis(4));
    set(handles.axis_zmin, 'String', new_axis(5));
    set(handles.axis_zmax, 'String', new_axis(6));

```

```

    rotate3d on

```

```

end

```

```

w_tog_update = 0;

```

```

% --- Executes on button press in mod_update.
function mod_update_Callback(hObject, eventdata, handles)
% hObject    handle to mod_update (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

global w_joint w_member w_constraint w_force w_mod_num

```

```

%
%
% search for specified feature
%
% is it Add / Modify / Remove ???
%
if (get(handles.mod_add,'Value')) == 1
    %can't update an add
    errordlg('To add a new feature, click "Apply" ','User Input Error');

elseif (get(handles.mod_mod,'Value')) == 1
    % search for legal feature and display
    %
    w_mod_num = str2double(get(handles.mod_num,'String'));

    % check for NaN
    if isnan(w_mod_num) == 1
        errordlg('You have entered an illegal feature number','User Input Error');
        return;
    end
    %

```

```

%check feature type
popup_sel_index = get(handles.listbox3, 'Value');
switch popup_sel_index
case 1
    %joint
    %check for valid entry, fill in user area
    if check_mod(w_joint, w_mod_num) == 0
        set(handles.user_x, 'String', w_joint(w_mod_num,1));
        set(handles.user_y, 'String', w_joint(w_mod_num,2));
        set(handles.user_z, 'String', w_joint(w_mod_num,3));
    end
case 2
    %member
    if check_mod(w_member, w_mod_num) == 0
        set(handles.user_x, 'String', w_member(w_mod_num,1));
        set(handles.user_y, 'String', w_member(w_mod_num,2));
        %set(handles.user_z, 'String', w_member(w_mod_num,3));
        set(handles.user_z, 'String', 'XXXXXXXXX');
    end
case 3
    %constraint

    c = check_mod2(w_constraint, w_mod_num);

    exist_val = c(1);
    index = c(2);

    if exist_val == 0
        set(handles.user_x, 'String', w_constraint(index,1));
        set(handles.user_y, 'String', w_constraint(index,2));
        set(handles.user_z, 'String', w_constraint(index,3));
    end
case 4
    %force
    c = check_mod2(w_force, w_mod_num);

    exist_val = c(1);
    index = c(2);

    if exist_val == 0
        set(handles.user_x, 'String', w_force(index,1));
        set(handles.user_y, 'String', w_force(index,2));
        set(handles.user_z, 'String', w_force(index,3));
    end
end

else
    % remove
    % display info and ask again
    errordlg('To remove a feature, click "Apply"', 'User Input Error');
end
end

```

```
%%
%%
%%
```

```
function feature = check_add(feature, x, y, z)
% For Joints and Members
%
% takes specified feature and checks for a repeat
% then adds the feature, if legal.
% internal use only, do not call
%global w_joint w_member w_constraint w_force
[a b] = size(feature);
repeat_val = 0;
for i = 1:a
    %check for repeat
    if feature(i,:) == [x y z]
        error('The feature values already exist','User Input Error');
        repeat_val = 1;
    end
end
%perform feature add if....
if repeat_val == 0
    feature(a+1,:) = [x y z];
end
return;
```

```
function feature = check_add2(feature, x, y, z, index)
% For Constraints and Forces
%
% takes specified feature and checks for a repeat
% then adds the feature, if legal.
% internal use only, do not call
%global w_joint w_member w_constraint w_force

[a b] = size(feature);
repeat_val = 0;

if isnan(feature) ~= 1

    for i = 1:a
        %check for repeat
        if feature(i,4) == index
            error('The feature values already exist','User Input Error');
            repeat_val = 1;
        end
    end
end
end
```

```

%perform feature add if....
if repeat_val == 0
    feature(a+1,:) = [x y z index];
end
return;

```

```

function exist_val = check_mod(feature, index)
% For Joints and Members
%
% takes specifed feature and checks for valid entry
% then displays the entry in the user entry area, if legal.
% internal use only, do not call

```

```

%does the index exist?
[a b] = size(feature);
exist_val = 0;
if index > a
    error('The feature entry does not exist','User Input Error');
    exist_val = 1;
end
return;

```

```

function c = check_mod2(feature, index)
% For Constraints and Forces
%
% takes specifed feature and checks for valid entry
% then displays the entry in the user entry area, if legal.
% internal use only, do not call

```

```

%X-val Y-val Z-val Joint #

```

```

%does the index exist?
[a b] = size(feature);
exist_val = 1;

```

```

for i = 1:a
    if feature(i, 4) == index
        % this entry exists!
        exist_val = 0;
        placement = i;

        c(1) = 0;
        c(1,2) = i;
    end
end
end

```



```

if exist_val == 1
    error('The feature entry does not exist','User Input Error');
    c = [1 0];

end
return;

function send_axis = draw_plot( tog_jointnum, tog_membernum, tog_constraint, tog_force, tog_grid,
    tog_axis_label, tog_axis, tog_view, all_data, reaction_force)

global w_joint w_member w_constraint w_force w_mod_num
% update plot
%axes(handles.axes1);
cla;
hold on

xlabel('X axis')
ylabel('Y axis')
zlabel('Z axis')

%draw grid
if tog_grid == 1
    grid on
else
    grid off
end

% draw nodes
x=w_joint(:,1);
y=w_joint(:,2);
z=w_joint(:,3);

plot3(x,y,z,'ys','MarkerEdgeColor','k','MarkerFaceColor','y','MarkerSize',8);

% draw members
[a b] = size(w_member);

for i = 1:a
    j1 = w_member(i,1);
    j2 = w_member(i,2);

    xm = [ w_joint(j1,1) w_joint(j2,1) ];
    ym = [ w_joint(j1,2) w_joint(j2,2) ];
    zm = [ w_joint(j1,3) w_joint(j2,3) ];

    if size(all_data) == [1,1]
        plot3(xm,ym,zm,'b-','LineWidth',2)
    end

    %draw member number

```

```

if tog_membernum == 1
    text(mean(xm),mean(ym),mean(zm),num2str(i),...
        'BackgroundColor',[.7 .3 .2],...
        'VerticalAlignment','bottom')
end

%
%
%      %draw output labels
%      if j == b & i <= m_length
%          label = num2str(i,2);
%
%          text(j-2,i,'M',...
%              'BackgroundColor',[.7 .3 .2],...
%              'EdgeColor','k',...
%              'HorizontalAlignment','center')
%
%          text(j-1.5,i,label,...
%              'BackgroundColor',[.7 .3 .2],...
%              'EdgeColor','k',...
%              'HorizontalAlignment','center',...
%              'FontWeight','bold')
%      elseif j == b
%          label = num2str(i-m_length,2);
%
%          text(j-2,i,'R',...
%              'BackgroundColor',[.6 .5 .4],...
%              'EdgeColor','k',...
%              'HorizontalAlignment','center')
%
%          text(j-1.5,i,label,...
%              'BackgroundColor',[.6 .5 .4],...
%              'EdgeColor','k',...
%              'HorizontalAlignment','center',...
%              'FontWeight','bold')
%
%      end
%

%draw analysis data
if size(all_data) ~= [1,1]

    %get scale
    react_scale = max(abs(reaction_force));

    line_scale = 2;

    if (reaction_force(i)/react_scale) > .9
        plot3(xm,ym,zm,'r-','LineWidth',5*line_scale)
    elseif (reaction_force(i)/react_scale) > .7
        plot3(xm,ym,zm,'r-','LineWidth',4*line_scale)
    elseif (reaction_force(i)/react_scale) > .4
        plot3(xm,ym,zm,'r-','LineWidth',2*line_scale)
    end
end

```

```

elseif (reaction_force(i)/react_scale) > .1
    plot3(xm,ym,zm,'r-','LineWidth',1*line_scale)

elseif (reaction_force(i)/react_scale) > 0
    plot3(xm,ym,zm,'r-','LineWidth',.5*line_scale)
elseif (reaction_force(i)/react_scale) == 0
    plot3(xm,ym,zm,'k-','LineWidth',line_scale)
elseif (reaction_force(i)/react_scale) < -.9
    plot3(xm,ym,zm,'b-','LineWidth',5*line_scale)
elseif (reaction_force(i)/react_scale) < -.7
    plot3(xm,ym,zm,'b-','LineWidth',4*line_scale)
elseif (reaction_force(i)/react_scale) < -.4
    plot3(xm,ym,zm,'b-','LineWidth',2*line_scale)
elseif (reaction_force(i)/react_scale) < -.1
    plot3(xm,ym,zm,'b-','LineWidth',1*line_scale)
else
    plot3(xm,ym,zm,'b-','LineWidth',.5*line_scale)

end

%redraw members as scaled to the weight

val = num2str(reaction_force(i),'%7.1f');

if tog_membernum == 1
    if reaction_force(i) <= 0
        text(mean(xm),mean(ym),mean(zm),val,...
            'BackgroundColor',[0 .5 .8],...
            'EdgeColor','k',...
            'HorizontalAlignment','center',...
            'VerticalAlignment','top')
    else
        text(mean(xm),mean(ym),mean(zm),val,...
            'BackgroundColor',[.8 0 0],...
            'EdgeColor','k',...
            'HorizontalAlignment','center',...
            'VerticalAlignment','top')
    end
end

end

end
end

```

```

%get axis info to scale force vector and constraints

user_axis = axis;

x_length = user_axis(2) - user_axis(1);
y_length = user_axis(4) - user_axis(3);

if length(user_axis) == 6
    z_length = user_axis(6) - user_axis(5);
else
    z_length = 0;
end

%scale graphics
scale_line = sqrt(x_length^2 + y_length^2 + z_length^2);

%draw force
if tog_force == 1
    if isnan(w_force) ~= 1
        %max force is 10% of axis
        max_size = 0.1;

        % the scale is equal
        % so take the largest value
        if size(all_data) == [1,1]
            scale_force = max(max(abs(w_force)));
        else
            scale_force = max([max(max(abs(w_force))) max(abs(reaction_force))]);
        end

        xf_scale = max_size*scale_line/scale_force;
        yf_scale = max_size*scale_line/scale_force;
        zf_scale = max_size*scale_line/scale_force;

    end

    %draw forces
    [a b] = size(w_force);
    for i = 1:a

        index = w_force(i,4);

        xf(1) = w_joint(index,1);
        yf(1) = w_joint(index,2);
        zf(1) = w_joint(index,3);

        xf(2) = xf(1) + w_force(i,1)*xf_scale;
        yf(2) = yf(1) + w_force(i,2)*yf_scale;
        zf(2) = zf(1) + w_force(i,3)*zf_scale;
    end
end

```

```

%draw FX
plot3([xf(1) xf(2)], [yf(1) yf(1)], [zf(1) zf(1)], 'r-', 'LineWidth', 1)

%draw FY
plot3([xf(1) xf(1)], [yf(1) yf(2)], [zf(1) zf(1)], 'r-', 'LineWidth', 1)

%draw FZ
plot3([xf(1) xf(1)], [yf(1) yf(1)], [zf(1) zf(2)], 'r-', 'LineWidth', 1)

%draw diagonal
plot3(xf, yf, zf, 'r-', 'LineWidth', 3)
plot3(xf(2), yf(2), zf(2), 'rd', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'r', 'MarkerSize', 10)

end
end

%create constraint scale

if tog_constraint == 1 & size(all_data) == [1,1]

    %max force is 5% of axis
    max_size = 0.05;

    xc_scale = max_size*scale_line;
    yc_scale = max_size*scale_line;
    zc_scale = max_size*scale_line;

    %draw constraints
    [a b] = size(w_constraint);

    for i = 1:a

        index = w_constraint(i,4);

        xc(1) = w_joint(index,1);
        yc(1) = w_joint(index,2);
        zc(1) = w_joint(index,3);

        xc(3) = w_joint(index,1) - w_constraint(i,1)*xc_scale;
        yc(3) = w_joint(index,2) - w_constraint(i,2)*yc_scale;
        zc(3) = w_joint(index,3) - w_constraint(i,3)*zc_scale;

        xc(2) = w_joint(index,1) + w_constraint(i,1)*xc_scale;
        yc(2) = w_joint(index,2) + w_constraint(i,2)*yc_scale;
        zc(2) = w_joint(index,3) + w_constraint(i,3)*zc_scale;

        %draw FX
        plot3([xc(3) xc(2)], [yc(1) yc(1)], [zc(1) zc(1)], 'go-', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'g')
        %draw FY
        plot3([xc(1) xc(1)], [yc(3) yc(2)], [zc(1) zc(1)], 'go-', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'g')
        %draw FZ

```

```

    plot3([xc(1) xc(1)], [yc(1) yc(1)], [zc(3) zc(2)], 'go-', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'g')
end

```

```

elseif tog_constraint == 1 & tog_force == 1

```

```

    [m_length b] = size(w_member);
    [rf_length b] = size(reaction_force);

```

```

    count = m_length+1;

```

```

    %draw constraints
    [a b] = size(w_constraint);

```

```

    for i = 1:a

```

```

        index = w_constraint(i,4);

```

```

        xf(1) = w_joint(index,1);
        yf(1) = w_joint(index,2);
        zf(1) = w_joint(index,3);

```

```

        xf(2) = xf(1);
        yf(2) = yf(1);
        zf(2) = zf(1);

```

```

        xm = [ w_joint(j1,1) w_joint(j2,1) ];
        ym = [ w_joint(j1,2) w_joint(j2,2) ];
        zm = [ w_joint(j1,3) w_joint(j2,3) ];

```

```

        if w_constraint(i,1) == 1
            xf(2) = xf(1) + reaction_force(count)*xf_scale;

```

```

        %draw FX

```

```

        plot3(xf, yf, zf, 'g-', 'LineWidth', 3)
        plot3( xf(2), yf(2), zf(2), ...
            'gd', 'MarkerEdgeColor', ...
            'k', 'MarkerFaceColor', 'g', ...
            'MarkerSize', 10)

```

```

        val = num2str(reaction_force(count), '%7.1f');

```

```

        if reaction_force(count) <= 0
            text(xf(2), yf(2), zf(2), val, ...
                'BackgroundColor', [0 .5 .8], ...

```

```

        'EdgeColor','k',...
        'HorizontalAlignment','center',...
        'VerticalAlignment','top')
    else
        text(xf(2), yf(2), zf(2),val,...
            'BackgroundColor',[.8 0 0],...
            'EdgeColor','k',...
            'HorizontalAlignment','center',...
            'VerticalAlignment','top')

    end

    count = count + 1;

end

xf(2) = xf(1);

if w_constraint(i,2) == 1
    yf(2) = yf(1) + reaction_force(count)*yf_scale;

    plot3(xf, yf, zf,'g-','LineWidth',3)
    plot3( xf(2), yf(2), zf(2),...
        'gd','MarkerEdgeColor',...
        'k','MarkerFaceColor','g',...
        'MarkerSize',10)

    val = num2str(reaction_force(count),'%7.1f');

    if reaction_force(count) <= 0
        text(xf(2), yf(2), zf(2),val,...
            'BackgroundColor',[0 .5 .8],...
            'EdgeColor','k',...
            'HorizontalAlignment','center',...
            'VerticalAlignment','top')
    else
        text(xf(2), yf(2), zf(2),val,...
            'BackgroundColor',[.8 0 0],...
            'EdgeColor','k',...
            'HorizontalAlignment','center',...
            'VerticalAlignment','top')

    end

    count = count + 1;

end

yf(2) = yf(1);

if w_constraint(i,3) == 1

```

```

zf(2) = zf(1) + reaction_force(count)*zf_scale;

%draw FZ
plot3(xf, yf, zf,'g-','LineWidth',3)
plot3( xf(2), yf(2), zf(2),...
      'gd','MarkerEdgeColor',...
      'k','MarkerFaceColor','g',...
      'MarkerSize',10)

val = num2str(reaction_force(count),'%7.1f');

if reaction_force(count) <= 0
    text(xf(2), yf(2), zf(2),val,...
        'BackgroundColor',[0 .5 .8],...
        'EdgeColor','k',...
        'HorizontalAlignment','center',...
        'VerticalAlignment','top')
else
    text(xf(2), yf(2), zf(2),val,...
        'BackgroundColor',[.8 0 0],...
        'EdgeColor','k',...
        'HorizontalAlignment','center',...
        'VerticalAlignment','top')

end

count = count + 1;
end

end

%draw reaction force

end

%set axis

if tog_view == 1
    axis equal
end

%get axis for graphics
new_axis = axis;

%offset 2%
max_size = 0.02;

```



```

xj_scale = max_size*x_length;
yj_scale = max_size*y_length;
zj_scale = max_size*z_length;

% draw joint number
if tog_jointnum == 1
    [a b] = size(w_joint);
    for i = 1:a
        x=w_joint(i,1) + xj_scale;
        y=w_joint(i,2) + yj_scale;
        z=w_joint(i,3) + zj_scale;

        text(x,y,z,num2str(i),'BackgroundColor',[0 .8 .8] )
    end
end

%get axis info for axis and labels
%draw origin
user_axis = axis;

if user_axis(1) > 0
    user_axis(1) = 0;
elseif user_axis(2) < 0
    user_axis(2) = 0;
end

if user_axis(3) > 0
    user_axis(3) = 0;
elseif user_axis(4) < 0
    user_axis(4) = 0;
end

if length(user_axis) == 6
    if user_axis(5) > 0
        user_axis(5) = 0;
    elseif user_axis(6) < 0
        user_axis(6) = 0;
    end
end

if tog_axis_label == 1

    %axis offset scale
    offset = 1.2;

    %draw X-X
    text(user_axis(1)*offset, 0,0,'-X ','BackgroundColor',[.7 .9 .7])
    text(user_axis(2)*offset, 0,0,'+X ','BackgroundColor',[.7 .9 .7])

    %draw Y-Y
    text(0, user_axis(3)*offset, 0, '-Y ','BackgroundColor',[.7 .9 .7])
    text(0, user_axis(4)*offset, 0, '+Y ','BackgroundColor',[.7 .9 .7])

```

```

%draw Z-Z
if length(user_axis) == 6
    text(0,0, user_axis(5)*offset, '-Z ', 'BackgroundColor',[.7 .9 .7])
    text(0,0, user_axis(6)*offset, '+Z ', 'BackgroundColor',[.7 .9 .7] )
end

end

if tog_axis == 1
    %draw X-X
    plot3([user_axis(1) user_axis(2)], [0 0],[0 0], 'k-', 'LineWidth',3)
    %draw Y-Y
    plot3([0 0], [user_axis(3) user_axis(4)], [0 0], 'k-', 'LineWidth',3)
    %draw Z-Z
    if length(user_axis) == 6
        plot3([0 0], [0 0],[user_axis(5) user_axis(6)], 'k-', 'LineWidth',3)
    end

end

% redraw, just in case
if tog_view == 1
    axis equal
end

%send axis
send_axis = axis;

return;

function feature = subtract_joint(feature, tog_member, mod_num);

% remove features connected to the joint
% lower joint number if needed

[a b] = size(feature);

if tog_member == 1
    %reset member
    for i = 1:a

        %by removing entris, the increment can be larger than the matrix
        if i <= a

            if feature(i,1) == mod_num | feature(i,2) == mod_num
                %feature exists on joint
                %remove feature row
                feature(i,:) = [];

                %resize a
                [a b] = size(feature);
            end
        end
    end
end

```

```

elseif feature(i,1) > mod_num | feature(i,2) > mod_num
    %joint needs adjusted

    if feature(i,1) > mod_num
        feature(i,1) = feature(i,1) - 1;
    end

    if feature(i,2) > mod_num
        feature(i,2) = feature(i,2) - 1;
    end
end
end
end

elseif tog_member == 0
    %reset constraint / force
    for i = 1:a

        %by removing entris, the increment can be larger than the matrix
        if i <= a
            if feature(i,4) == mod_num
                feature(i,:) = [];

                %resize a
                [a b] = size(feature);

                elseif feature(i,4) > mod_num
                    feature(i,4) = feature(i,4) - 1;
                end
            end
        end
    end
end

return;

```

```

% --- Executes during object creation, after setting all properties.
function axis_xmin_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axis_xmin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function axis_xmin_Callback(hObject, eventdata, handles)
% hObject    handle to axis_xmin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of axis_xmin as text
%       str2double(get(hObject,'String')) returns contents of axis_xmin as a double

% --- Executes during object creation, after setting all properties.
function axis_xmax_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axis_xmax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function axis_xmax_Callback(hObject, eventdata, handles)
% hObject    handle to axis_xmax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```
% Hints: get(hObject,'String') returns contents of axis_xmax as text
%      str2double(get(hObject,'String')) returns contents of axis_xmax as a double
```

```
% --- Executes during object creation, after setting all properties.
function axis_ymin_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axis_ymin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%      See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function axis_ymin_Callback(hObject, eventdata, handles)
% hObject    handle to axis_ymin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of axis_ymin as text
%      str2double(get(hObject,'String')) returns contents of axis_ymin as a double
```

```
% --- Executes during object creation, after setting all properties.
function axis_ymax_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axis_ymax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%      See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function axis_ymax_Callback(hObject, eventdata, handles)
% hObject    handle to axis_ymax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of axis_ymax as text
%      str2double(get(hObject,'String')) returns contents of axis_ymax as a double
```

```

% --- Executes during object creation, after setting all properties.
function axis_zmin_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axis_zmin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function axis_zmin_Callback(hObject, eventdata, handles)
% hObject    handle to axis_zmin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of axis_zmin as text
%        str2double(get(hObject,'String')) returns contents of axis_zmin as a double

% --- Executes during object creation, after setting all properties.
function axis_zmax_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axis_zmax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function axis_zmax_Callback(hObject, eventdata, handles)
% hObject    handle to axis_zmax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of axis_zmax as text
%        str2double(get(hObject,'String')) returns contents of axis_zmax as a double

% --- Executes during object creation, after setting all properties.

```

```

function edit40_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit40 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit40_Callback(hObject, eventdata, handles)
% hObject    handle to edit40 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit40 as text
%       str2double(get(hObject,'String')) returns contents of edit40 as a double

% --- Executes on button press in axis_update.
function axis_update_Callback(hObject, eventdata, handles)
% hObject    handle to axis_update (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global w_joint w_all_data w_reaction_force

if isnan(w_joint) ~= 1

    %get axis data and modify
    xmin = str2double(get(handles.axis_xmin,'String'));
    xmax = str2double(get(handles.axis_xmax,'String'));

    ymin = str2double(get(handles.axis_ymin,'String'));
    ymax = str2double(get(handles.axis_ymax,'String'));

    zmin = str2double(get(handles.axis_zmin,'String'));
    zmax = str2double(get(handles.axis_zmax,'String'));

    axes(handles.axes1);

    axis([xmin xmax ymin ymax zmin zmax]);

    switch get(handles.ShowJointMenu,'Checked')
        case 'on'
            tog_jointnum = 1;
        case 'off'

```

```

        tog_jointnum = 0;
    end

    switch get(handles.ShowMemberMenu,'Checked')
        case 'on'
            tog_membernum = 1;
        case 'off'
            tog_membernum = 0;
        end

    switch get(handles.ShowConstraintMenu,'Checked')
        case 'on'
            tog_constraint = 1;
        case 'off'
            tog_constraint = 0;
        end

    switch get(handles.ShowForceMenu,'Checked')
        case 'on'
            tog_force = 1;
        case 'off'
            tog_force = 0;
        end

    switch get(handles.ShowGridMenu,'Checked')
        case 'on'
            tog_grid = 1;
        case 'off'
            tog_grid = 0;
        end

    switch get(handles.ShowAxisLabelMenu,'Checked')
        case 'on'
            tog_axis_label = 1;
        case 'off'
            tog_axis_label = 0;
        end

    switch get(handles.ShowAxesMenu,'Checked')
        case 'on'
            tog_axes = 1;
        case 'off'
            tog_axes = 0;
        end

    %reset view, no
    tog_view = 0;

    %check for analysis, draw
    if sum(w_all_data) == 0

        all_data = 1;
    end

```



```

        reaction_force = 1;
    else

        all_data = w_all_data;
        reaction_force = w_reaction_force;

    end

    axes(handles.axes1);
    new_axis = draw_plot( tog_jointnum, tog_membernum, tog_constraint, tog_force, tog_grid,
tog_axis_label, tog_axes, tog_view, all_data, reaction_force);
    rotate3d on

else
    errordlg('No model exists. Add a Joint from the "Feature Type" menu.','No Model Available')
end

% --- Executes during object creation, after setting all properties.
function edit42_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit42 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit42_Callback(hObject, eventdata, handles)
% hObject    handle to edit42 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit42 as text
%       str2double(get(hObject,'String')) returns contents of edit42 as a double

% --- Executes on button press in axis_reset.
function axis_reset_Callback(hObject, eventdata, handles)
% hObject    handle to axis_reset (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global w_joint w_all_data w_reaction_force

axes(handles.axes1);
cla;

```

```

hold off

axis equal
hold on
%axis([0 1 0 1 0 1])

if isnan(w_joint) ~= 1

    switch get(handles.ShowJointMenu,'Checked')
        case 'on'
            tog_jointnum = 1;
        case 'off'
            tog_jointnum = 0;
        end

    switch get(handles.ShowMemberMenu,'Checked')
        case 'on'
            tog_membernum = 1;
        case 'off'
            tog_membernum = 0;
        end

    switch get(handles.ShowConstraintMenu,'Checked')
        case 'on'
            tog_constraint = 1;
        case 'off'
            tog_constraint = 0;
        end

    switch get(handles.ShowForceMenu,'Checked')
        case 'on'
            tog_force = 1;
        case 'off'
            tog_force = 0;
        end

    switch get(handles.ShowGridMenu,'Checked')
        case 'on'
            tog_grid = 1;
        case 'off'
            tog_grid = 0;
        end

    switch get(handles.ShowAxisLabelMenu,'Checked')
        case 'on'
            tog_axis_label = 1;
        case 'off'
            tog_axis_label = 0;
        end

    switch get(handles.ShowAxesMenu,'Checked')
        case 'on'

```

```

        tog_axes = 1;
    case 'off'
        tog_axes = 0;
    end

    %reset view, no
    tog_view = 1;

    %also reset analysis
    w_all_data = [];
    w_reaction_force = [];

    %
    %   %check for analysis, draw
    %   if size(w_all_data) == [1,1]
    %
    %       all_data = 1;
    %       reaction_force = 1;
    %   else
    %
    %       all_data = w_all_data;
    %       reaction_force = w_reaction_force;
    %
    %   end

    axes(handles.axes1);
    new_axis = draw_plot( tog_jointnum, tog_membernum, tog_constraint, tog_force, tog_grid,
    tog_axis_label, tog_axes, tog_view, 1, 1);
    rotate3d on

    %rotate3d on
    new_axis = axis;
    %new_axis = round(axis);

    for i = 1:length(new_axis)

        new_axis(i) = round(new_axis(i)*10)/10;

    end

    axis(new_axis);

    set(handles.axis_xmin, 'String', new_axis(1));
    set(handles.axis_xmax, 'String', new_axis(2));
    set(handles.axis_ymin, 'String', new_axis(3));
    set(handles.axis_ymax, 'String', new_axis(4));

    if length(new_axis) == 6
        set(handles.axis_zmin, 'String', new_axis(5));
        set(handles.axis_zmax, 'String', new_axis(6));
    end

    rotate3d on

```

```

%set view
az = 30;
el = 30;
view(az, el);

else
    errordlg('No model exists. Add a Joint from the "Feature Type" menu.','No Model Available')
end

% -----
function ShowAxisLabelMenu_Callback(hObject, eventdata, handles)
% hObject    handle to ShowAxisLabelMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

switch get(handles.ShowAxisLabelMenu,'Checked')
    case 'on'
        set(handles.ShowAxisLabelMenu, 'Checked', 'off')
    case 'off'
        set(handles.ShowAxisLabelMenu, 'Checked', 'on')
end

switch get(handles.ShowJointMenu,'Checked')
    case 'on'
        tog_jointnum = 1;
    case 'off'
        tog_jointnum = 0;
end

switch get(handles.ShowMemberMenu,'Checked')
    case 'on'
        tog_membernum = 1;
    case 'off'
        tog_membernum = 0;
end

switch get(handles.ShowConstraintMenu,'Checked')
    case 'on'
        tog_constraint = 1;
    case 'off'
        tog_constraint = 0;
end

switch get(handles.ShowForceMenu,'Checked')
    case 'on'
        tog_force = 1;
    case 'off'
        tog_force = 0;
end

```

```

switch get(handles.ShowGridMenu,'Checked')
case 'on'
    tog_grid = 1;
case 'off'
    tog_grid = 0;
end

switch get(handles.ShowAxisLabelMenu,'Checked')
case 'on'
    tog_axis_label = 1;
case 'off'
    tog_axis_label = 0;
end

switch get(handles.ShowAxesMenu,'Checked')
case 'on'
    tog_axes = 1;
case 'off'
    tog_axes = 0;
end

%reset view
tog_view = 0;

global w_all_data w_reaction_force

%check for analysis, draw
if sum(w_all_data) == 0

    all_data = 1;
    reaction_force = 1;
else

    all_data = w_all_data;
    reaction_force = w_reaction_force;

end

axes(handles.axes1);
new_axis = draw_plot( tog_jointnum, tog_membernum, tog_constraint, tog_force, tog_grid,
tog_axis_label, tog_axes, tog_view, all_data, reaction_force);
rotate3d on

% -----
function ShowAxesMenu_Callback(hObject, eventdata, handles)
% hObject    handle to ShowAxesMenu (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

switch get(handles.ShowAxesMenu,'Checked')
case 'on'

```

```

        set(handles.ShowAxesMenu, 'Checked', 'off')
    case 'off'
        set(handles.ShowAxesMenu, 'Checked', 'on')
    end

    switch get(handles.ShowJointMenu, 'Checked')
    case 'on'
        tog_jointnum = 1;
    case 'off'
        tog_jointnum = 0;
    end

    switch get(handles.ShowMemberMenu, 'Checked')
    case 'on'
        tog_membernum = 1;
    case 'off'
        tog_membernum = 0;
    end

    switch get(handles.ShowConstraintMenu, 'Checked')
    case 'on'
        tog_constraint = 1;
    case 'off'
        tog_constraint = 0;
    end

    switch get(handles.ShowForceMenu, 'Checked')
    case 'on'
        tog_force = 1;
    case 'off'
        tog_force = 0;
    end

    switch get(handles.ShowGridMenu, 'Checked')
    case 'on'
        tog_grid = 1;
    case 'off'
        tog_grid = 0;
    end

    switch get(handles.ShowAxisLabelMenu, 'Checked')
    case 'on'
        tog_axis_label = 1;
    case 'off'
        tog_axis_label = 0;
    end

    switch get(handles.ShowAxesMenu, 'Checked')
    case 'on'
        tog_axes = 1;
    case 'off'
        tog_axes = 0;
    end
end

```

```

%reset view
tog_view = 0;

global w_all_data w_reaction_force

%check for analysis, draw
if sum(w_all_data) == 0

    all_data = 1;
    reaction_force = 1;
else

    all_data = w_all_data;
    reaction_force = w_reaction_force;

end

axes(handles.axes1);
new_axis = draw_plot( tog_jointnum, tog_membernum, tog_constraint, tog_force, tog_grid,
tog_axis_label, tog_axes, tog_view, all_data, reaction_force);
rotate3d on

% -----
function NewMenu_Callback(hObject, eventdata, handles)
% hObject    handle to NewMenu (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global w_joint
clc
[a b] = size(w_joint);

if a ~= 0
    button = questdlg('Do you want to save the model',...
'Start New File?','Yes','No','No');
    if strcmp(button,'Yes')

        [file, path] = uiputfile({'*.mat','Matlab Model File (*.mat)'},'Save As');

        if ~isequal(file, 0)
            %open(file);

            filename = char(file);
            %load file
            save(filename);

        end
    end
end

```

```

elseif strcmp(button,'No')
    %
    axes(handles.axes1);
    cla;

    clear global

    n = round(rand*50+20);
    m = rand*3 + 1;

    q = m*rand(n,n)+(peaks(n));

    surf(q);
    axis equal
end
else
    axes(handles.axes1);
    cla;

    clear global

    n = round(rand*50+20);
    m = rand*3 + 1;

    q = m*rand(n,n)+(peaks(n));

    surf(q);
    axis equal

end

```

```

% -----
function SaveMenu_Callback(hObject, eventdata, handles)
% hObject    handle to SaveMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global w_joint w_member w_constraint w_force

% prompt = {'Enter filename (default is *.fea)'};
% dlg_title = 'Save Project';
% num_lines= 1;
% def      = {'project1.fea'};

```



```

% answer = inputdlg(prompt,dlg_title,num_lines,def);
%
% n = 'aaa';%answer(1,1)
%
% save 'aaa.mat' w_joint w_member w_constraint w_force

[file, path] = uiputfile({'*.mat','Matlab Model File (*.mat)'},'Save As');

if ~isequal(file, 0)

    filename = char(file);
    %load file
    save(filename);

end

% --- Executes during object creation, after setting all properties.
function edit44_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit44 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit44_Callback(hObject, eventdata, handles)
% hObject    handle to edit44 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit44 as text
%       str2double(get(hObject,'String')) returns contents of edit44 as a double

% --- Executes during object creation, after setting all properties.
function edit46_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit46 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.

```

```

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

```

```

function edit46_Callback(hObject, eventdata, handles)
% hObject    handle to edit46 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit46 as text
%        str2double(get(hObject,'String')) returns contents of edit46 as a double

```

```

% --- Executes on button press in pushbutton12.
function pushbutton12_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

% --- Executes on button press in user_2d.
function user_2d_Callback(hObject, eventdata, handles)
% hObject    handle to user_2d (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

% Hint: get(hObject,'Value') returns toggle state of user_2d
set(handles.user_2d, 'Value', 1);
set(handles.user_3d, 'Value', 0);

set(handles.user_2dxy, 'Value', 0);
set(handles.user_2dyz, 'Value', 0);
set(handles.user_2dxz, 'Value', 1);

```

```

% --- Executes on button press in user_3d.
function user_3d_Callback(hObject, eventdata, handles)
% hObject    handle to user_3d (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

% Hint: get(hObject,'Value') returns toggle state of user_3d

set(handles.user_2d, 'Value', 0);
set(handles.user_3d, 'Value', 1);

set(handles.user_2dxy, 'Value', 0);
set(handles.user_2dyz, 'Value', 0);

```

```

set(handles.user_2dxz, 'Value', 0);

% --- Executes on button press in user_2dxy.
function user_2dxy_Callback(hObject, eventdata, handles)
% hObject    handle to user_2dxy (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

if get(handles.user_2d, 'Value') == 1

    set(handles.user_2dxy, 'Value', 1);
    set(handles.user_2dyz, 'Value', 0);
    set(handles.user_2dxz, 'Value', 0);
else
    set(handles.user_2dxy, 'Value', 0);
    set(handles.user_2dyz, 'Value', 0);
    set(handles.user_2dxz, 'Value', 0);
    beep
end

% Hint: get(hObject,'Value') returns toggle state of user_2dxy

% --- Executes on button press in user_2dxz.
function user_2dxz_Callback(hObject, eventdata, handles)
% hObject    handle to user_2dxz (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of user_2dxz
if get(handles.user_2d, 'Value') == 1

    set(handles.user_2dxy, 'Value', 0);
    set(handles.user_2dyz, 'Value', 0);
    set(handles.user_2dxz, 'Value', 1);
else
    set(handles.user_2dxy, 'Value', 0);
    set(handles.user_2dyz, 'Value', 0);
    set(handles.user_2dxz, 'Value', 0);
    beep
end

% --- Executes on button press in user_2dyz.
function user_2dyz_Callback(hObject, eventdata, handles)
% hObject    handle to user_2dyz (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of user_2dyz
if get(handles.user_2d, 'Value') == 1

    set(handles.user_2dxy, 'Value', 0);

```

```

    set(handles.user_2dyz, 'Value', 1);
    set(handles.user_2dxz, 'Value', 0);
else
    set(handles.user_2dxy, 'Value', 0);
    set(handles.user_2dyz, 'Value', 0);
    set(handles.user_2dxz, 'Value', 0);
    beep
end

% -----
function ShowToggleMenu_Callback(hObject, eventdata, handles)
% hObject    handle to ShowToggleMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

tog_count = 0;

switch get(handles.ShowJointMenu,'Checked')
case 'on'
    tog_count = tog_count + 1;
end

switch get(handles.ShowMemberMenu,'Checked')
case 'on'
    tog_count = tog_count + 1;
end

switch get(handles.ShowConstraintMenu,'Checked')
case 'on'
    tog_count = tog_count + 1;
end

switch get(handles.ShowForceMenu,'Checked')
case 'on'
    tog_count = tog_count + 1;
end

switch get(handles.ShowGridMenu,'Checked')
case 'on'
    tog_grid = 1;
case 'off'
    tog_grid = 0;
end

switch get(handles.ShowAxisLabelMenu,'Checked')
case 'on'
    tog_axis_label = 1;
case 'off'
    tog_axis_label = 0;
end

switch get(handles.ShowAxesMenu,'Checked')
case 'on'

```

```

    tog_count = tog_count + 1;
end

%reset view
tog_view = 0;

axes(handles.axes1);

global w_all_data w_reaction_force

%check for analysis, draw

if sum(w_all_data) == 0

    all_data = 1;
    reaction_force = 1;
else

    all_data = w_all_data;
    reaction_force = w_reaction_force;

end

if tog_count > 2
    new_axis = draw_plot( 0, 0, 0, 0, tog_grid, 0, 0, tog_view, all_data, reaction_force);
    set(handles.ShowJointMenu, 'Checked', 'off')
    set(handles.ShowMemberMenu, 'Checked', 'off')
    set(handles.ShowConstraintMenu, 'Checked', 'off')
    set(handles.ShowForceMenu, 'Checked', 'off')

    set(handles.ShowAxesMenu, 'Checked', 'off')
    set(handles.ShowAxisLabelMenu, 'Checked', 'off')

else
    new_axis = draw_plot( 1, 1, 1, 1, tog_grid, tog_axis_label, 1, tog_view, all_data, reaction_force);
    set(handles.ShowJointMenu, 'Checked', 'on')
    set(handles.ShowMemberMenu, 'Checked', 'on')
    set(handles.ShowConstraintMenu, 'Checked', 'on')
    set(handles.ShowForceMenu, 'Checked', 'on')

    set(handles.ShowAxesMenu, 'Checked', 'on')
    %set(handles.ShowAxisLabelMenu, 'Checked', 'on')

end
rotate3d on

function draw_value(a,b,matrix)

% global w_member
%
% [m_length m_width] = size(w_member);

```

```

%
% hold on
% axis([1 b 1 a])
%
% grid on
%
% %
% for i = 1:a
%     for j = 1:b
%         if matrix(i,j) ~= 0 | j == b-4 | j == b
%
%             %draw output labels
%             if j == b & i <= m_length
%                 label = num2str(i,2);
%
%                 text(j-2,i,'M',...
%                     'BackgroundColor',[.7 .3 .2],...
%                     'EdgeColor','k',...
%                     'HorizontalAlignment','center')
%
%                 text(j-1.5,i,label,...
%                     'BackgroundColor',[.7 .3 .2],...
%                     'EdgeColor','k',...
%                     'HorizontalAlignment','center',...
%                     'FontWeight','bold')
%             elseif j == b
%                 label = num2str(i-m_length,2);
%
%                 text(j-2,i,'R',...
%                     'BackgroundColor',[.6 .5 .4],...
%                     'EdgeColor','k',...
%                     'HorizontalAlignment','center')
%
%                 text(j-1.5,i,label,...
%                     'BackgroundColor',[.6 .5 .4],...
%                     'EdgeColor','k',...
%                     'HorizontalAlignment','center',...
%                     'FontWeight','bold')
%
%             end
%
%         %draw box around matrix
%         plot([b-5.5 b-5.5 b-5.6 b-5.6 b-5.5],[1 a a 1 1])
%
%         plot([b-2.5 b-2.5 b-2.6 b-2.6 b-2.5],[1 a a 1 1])
%
%     %draw value
%     if abs(matrix(i,j)) <= 1
%         val = num2str(matrix(i,j), 2);
%     else
%         val = num2str(matrix(i,j), '%6.1f');

```

```

%     end
%
%
%     if matrix(i,j) <= 0
%         text(j,i,val,...
%             'BackgroundColor',[0 .5 .8],...
%             'EdgeColor','k',...
%             'HorizontalAlignment','center')
%     else
%         text(j,i,val,...
%             'BackgroundColor',[.8 0 0],...
%             'EdgeColor','k',...
%             'HorizontalAlignment','center')
%     end
% end
% end
% end
% return;

function analysis_scale = draw_analysis(all_data, force_data, reaction_force)

global w_joint, w_member, w_constraint, w_force
%
%
% send analysis info to draw area

% get w_member values

% get reaction forces

% draw reaction forces

% draw member values

% send data back to control panel

% -----
function Error_Plane_Callback(hObject, eventdata, handles)
% hObject    handle to Error_Plane (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

global w_joint w_member w_constraint w_force w_all_data w_reaction_force w_force_data

%1. build matrix from members / joints
%2. apply forces then constraints

%verify nothing exists on a removed column

[j_length b] = size(w_joint);
[m_length b] = size(w_member);

all_data = zeros(j_length *3, m_length);

%build matrix

for i = 1:j_length

    %cycle through each joint and find connectivity
    for j = 1:m_length

        xyz1 = 0;
        xyz1 = 0;

        %if member uses joint i
        if (w_member(j,1) == i | w_member(j,2) == i)

            %member data
            j1 = w_member(j,1);
            j2 = w_member(j,2);

            if j1 == i

                xyz1 = w_joint(j1,:);
                xyz2 = w_joint(j2,:);

                other_j = j2;

            elseif j2 == i

                xyz1 = w_joint(j2,:);
                xyz2 = w_joint(j1,:);

                other_j = j1;

            end

            %get lengths
            all_length = xyz2 - xyz1;

            scalar_length = sqrt(sum( all_length.^2));

```



```

        % adjust to member columns
        % x portion
        all_data(i*3-2, j) = all_length(1) / scalar_length;
        % y portion
        all_data(i*3-1, j) = all_length(2) / scalar_length;
        % z portion
        all_data(i*3-0, j) = all_length(3) / scalar_length;

    end
    %elseif anything?

    %for loops
end
end

[a b] = size(all_data);

force_data = zeros(j_length * 3, 1);

%create force vector
[a b] = size(w_force);

for i = 1:a

    j1 = w_force(i,4);

    if w_force(i,1) ~= 0
        force_data(j1*3-2, 1) = w_force(i,1);
    end

    if w_force(i,2) ~= 0
        force_data(j1*3-1, 1) = w_force(i,2);
    end

    if w_force(i,3) ~= 0
        force_data(j1*3-0, 1) = w_force(i,3);
    end

end

%get joints with constraints

%move through the matrix backwards.
%this way the numbering scheme is easier to control
[a b] = size(w_constraint);

[ad_size b] = size(all_data);
[f_length b2] = size(force_data);

%how big does the matrix need to be?
mod_size = 0;

for i = 1:a

```

```

j1 = w_constraint(i,4);

% x constraint
if w_constraint(i,1) == 1

    mod_size = mod_size+1;

    all_data(j1*3-2,b+mod_size) = 1;
end

% y constraint
if w_constraint(i,2) == 1

    mod_size = mod_size+1;

    all_data(j1*3-1,b+mod_size) = 1;
end

% z constraint
if w_constraint(i,3) == 1

    mod_size = mod_size+1;

    all_data(j1*3-0,b+mod_size) = 1;
end
end

err = 0;
% %condition data for 3d / 2d
if get(handles.user_2d, 'Value') == 1
    %get 2d plane

    if get(handles.user_2dxy, 'Value') == 1
        offset = 0;
    elseif get(handles.user_2dxz, 'Value') == 1
        offset = 1;
    elseif get(handles.user_2dyz, 'Value') == 1
        offset = 2;
    end

    for i = j_length*3:-3:3
        % check joints on plane
        if sum(all_data(i-offset,:)) ~= 0 & err ~= 1
            if offset == 0
                errordlg('A joint exists with a Z coordinate other than zero. If the entire model is along the same
                XY Plane, then the results will be accurate. This could be an incorrect constraint','2D Plane Error')
            elseif offset == 1
                errordlg('A joint exists with a Y coordinate other than zero. If the entire model is along the same
                XZ Plane, then the results will be accurate. This could be an incorrect constraint','2D Plane Error')
            end
        end
    end
end

```

```

        else
            errordlg('A joint exists with a X coordinate other than zero. If the entire model is along the same
YZ Plane, then the results will be accurate. This could be an incorrect constraint','2D Plane Error')
        end
        err = 1;

    end

    %check forces on plane
    if sum(force_data(i-offset,:)) ~= 0 & err ~= 1
        if offset == 0
            errordlg('A force exists with a Z coordinate other than zero. The force will be ignored during
analysis','2D Plane Error')
        elseif offset == 1
            errordlg('A force exists with a Y coordinate other than zero. The force will be ignored during
analysis','2D Plane Error')
        else
            errordlg('A force exists with a X coordinate other than zero. The force will be ignored during
analysis','2D Plane Error')
        end

        err = 1;

    end

end

end

%check constraints

% %condition data for 3d / 2d
if get(handles.user_2d, 'Value') == 1
    %get 2d plane

    if get(handles.user_2dxy, 'Value') == 1
        if sum(w_constraint(:,3)) ~= 0
            errordlg('A constraint exists with a Z coordinate other than zero. This may cause a failure in
analysis','2D Plane Error')
            err = 1;
        end
        elseif get(handles.user_2dxz, 'Value') == 1
            if sum(w_constraint(:,2)) ~= 0
                errordlg('A constraint exists with a Y coordinate other than zero. This may cause a failure in
analysis','2D Plane Error')
                err = 1;
            end
            elseif get(handles.user_2dyz, 'Value') == 1
                if sum(w_constraint(:,1)) ~= 0
                    errordlg('A constraint exists with a X coordinate other than zero. This may cause a failure in
analysis','2D Plane Error')
                    err = 1;
                end
            end
        end
    end
end

```

```

end

if err == 0

    errordlg('No Error Detected', 'Error Checking Successful')
end

% -----
function Error_Constraint_Callback(hObject, eventdata, handles)
% hObject    handle to Error_Constraint (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global w_constraint

con_count = sum(w_constraint(:,1) + w_constraint(:,2) + w_constraint(:,3));

err = 0;
% %condition data for 3d / 2d
if get(handles.user_2d, 'Value') == 1

    if con_count ~= 3
        errordlg('2D models must have 3 constraints to solve the matrix', 'User Input Error')
        err = 1;
    end

else
    if con_count ~= 6
        errordlg('3D models must have 6 constraints to solve the matrix', 'User Input Error')
        err = 1;
    end

end

if err == 0
    errordlg('No Error Detected', 'Error Constraint Check')
end

% -----
function Error_Recovery_Callback(hObject, eventdata, handles)
% hObject    handle to Error_Recovery (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% sweep through the member, constraint, and force to verify a joint exists
% for each entry

```

```

button = questdlg('Search for invalid joint entries?',...
    'Scan for Joint and Repair','Yes','No','No');

if strcmp(button,'Yes')

    global w_joint w_member w_constraint w_force

    %get size of w_joint
    [joint_max b] = size(w_joint);

    %scan through w_member for connectivity
    [mem b] = size(w_member);
    for i = 1:mem

        if w_member(i,1) > joint_max | w_member(i,2) > joint_max
            %then remove feature
            w_member(i,:) = [];
        end
    end

    %scan through w_constraint
    [con b] = size(w_constraint);

    for i = 1:con

        if w_constraint(i,4) > joint_max
            %then remove feature
            w_constraint(i,:) = [];
        end
    end

    %scan through w_force
    [force_val b] = size(w_force);

    for i = 1:force_val

        if w_force(i,4) > joint_max
            %then remove feature
            w_force(i,:) = [];
        end
    end

    switch get(handles.ShowJointMenu,'Checked')
        case 'on'
            tog_jointnum = 1;
        case 'off'
            tog_jointnum = 0;
    end

    switch get(handles.ShowMemberMenu,'Checked')
        case 'on'

```

```

        tog_membernum = 1;
    case 'off'
        tog_membernum = 0;
    end

    switch get(handles.ShowConstraintMenu,'Checked')
        case 'on'
            tog_constraint = 1;
        case 'off'
            tog_constraint = 0;
        end

    switch get(handles.ShowForceMenu,'Checked')
        case 'on'
            tog_force = 1;
        case 'off'
            tog_force = 0;
        end

    switch get(handles.ShowGridMenu,'Checked')
        case 'on'
            tog_grid = 1;
        case 'off'
            tog_grid = 0;
        end

    switch get(handles.ShowAxisLabelMenu,'Checked')
        case 'on'
            tog_axis_label = 1;
        case 'off'
            tog_axis_label = 0;
        end

    switch get(handles.ShowAxesMenu,'Checked')
        case 'on'
            tog_axes = 1;
        case 'off'
            tog_axes = 0;
        end

    %reset view, no
    tog_view = 0;

    global w_all_data w_reaction_force

    %check for analysis, draw
    if sum(w_all_data) == 0

        all_data = 1;
        reaction_force = 1;
    else

        all_data = w_all_data;

```

```

        reaction_force = w_reaction_force;

    end

    axes(handles.axes1);
    new_axis = draw_plot( tog_jointnum, tog_membernum, tog_constraint, tog_force, tog_grid,
        tog_axis_label, tog_axes, tog_view, all_data, reaction_force);
    rotate3d on

    msgbox('Scan Complete')
end

% --- Executes on button press in check_drawFBD.
function check_drawFBD_Callback(hObject, eventdata, handles)
% hObject    handle to check_drawFBD (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of check_drawFBD

```

Appendix B.2

Ramp and Spring Energy Dynamics code

EFD_RAMP

```
function varargout = efd_ramp(varargin)
% EFD_RAMP M-file for efd_ramp.fig
%   EFD_RAMP, by itself, creates a new EFD_RAMP or raises the existing
%   singleton*.
%
%   H = EFD_RAMP returns the handle to a new EFD_RAMP or the handle to
%   the existing singleton*.
%
%   EFD_RAMP('CALLBACK',hObject,eventData,handles,...) calls the local
%   function named CALLBACK in EFD_RAMP.M with the given input arguments.
%
%   EFD_RAMP('Property','Value',...) creates a new EFD_RAMP or raises the
%   existing singleton*. Starting from the left, property value pairs are
%   applied to the GUI before efd_ramp_OpeningFunction gets called. An
%   unrecognized property name or invalid value makes property application
%   stop. All inputs are passed to efd_ramp_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
%   instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help efd_ramp

% Last Modified by GUIDE v2.5 30-May-2003 15:39:27

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',   gui_Singleton, ...
                  'gui_OpeningFcn', @efd_ramp_OpeningFcn, ...
                  'gui_OutputFcn',  @efd_ramp_OutputFcn, ...
                  'gui_LayoutFcn',  [] , ...
                  'gui_Callback',    []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
```



```

% End initialization code - DO NOT EDIT

% --- Executes just before efd_ramp is made visible.
function efd_ramp_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)
% varargin    command line arguments to efd_ramp (see VARARGIN)

% Choose default command line output for efd_ramp
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% This sets up the initial plot - only do when we are invisible
% so window can get raised using efd_ramp.
if strcmp(get(hObject,'Visible'),'off')
    surf(peaks);
end

%set initial data
set(handles.drag_no, 'Value', 1);
set(handles.ke_check, 'Value', 1);
set(handles.UnitsEnglish, 'Checked', 'off');
set(handles.UnitsMetric, 'Checked', 'on');

% UIWAIT makes efd_ramp wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = efd_ramp_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

%get output info
if (get(handles.figure_check,'Value')) == 1
    figure(1)
    clf reset;
else

```

```

    axes(handles.axes1);
    cla;
end

%determine units
switch get(handles.UnitsEnglish,'Checked')
    case 'on'
        unit_position = 'feet';
        unit_velocity = 'feet / second';
    case 'off'
        unit_position = 'meters';
        unit_velocity = 'meters / second';
end

%set view
view(2)

clc

%determine units
switch get(handles.UnitsEnglish,'Checked')
    case 'on'
        gravity = -32.2;
        unit_position = 'feet';
    case 'off'
        gravity = -9.81;%02;
        unit_position = 'meters';
end

%retrieve data
dt = str2double(get(handles.ti_dt,'String'));

%get KE data
ke_vmag = str2double(get(handles.ke_vmag,'String'));
ke_angle = str2double(get(handles.ke_angle,'String'));
ke_height = str2double(get(handles.ke_height,'String'));
ke_dy = str2double(get(handles.ke_dy,'String'));
ke_mass = str2double(get(handles.ke_mass,'String'));
ke_loss = str2double(get(handles.ke_loss,'String'));

%make ke_loss a percentage
ke_loss = ke_loss / 100;

%get Target data
td_xmin = str2double(get(handles.td_xmin,'String'));
td_xmax = str2double(get(handles.td_xmax,'String'));
td_ymin = str2double(get(handles.td_ymin,'String'));
td_ymax = str2double(get(handles.td_ymax,'String'));

```

```

%
% Energy From Ramp
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

KE = 0;
%calc total energy from initial motion KE
if (get(handles.ke_check,'Value')) == 1

    KE = (0.5 * ke_mass * ke_vmag ^ 2 + gravity * ke_mass * ke_dy) * (1-ke_loss);

    %
    %
    %verify Ramp energy
    %
    %inital velocity check for overcoming ramp height
    %only check if PE is turned off
    if KE < 0 & (get(handles.pe_check,'Value')) == 0
        errordlg('The inital velocity cannot overcome the ramp','Invalid Analysis')
        return;
    end

    % check ke_dy and ke_angle compliance
    if ke_angle > 360
        errordlg('The Ramp Angle is above 360 degrees','Invalid Analysis')
        return;
    end

    %check validity of data
    if ke_angle > 180 & ke_angle <= 360 & ke_dy > 0
        errordlg('The "Ramp Angle" and "dy" don"t correspond!!! For angles between 180 and 360 degress,
"dy" must be negative. Please modify the "Ramp Angle" or "dy". ','Invalid Analysis')
        return;

    elseif ke_angle > 0 & ke_angle <= 180 & ke_dy < 0
        errordlg('The "Ramp Angle" and "dy" don"t correspond!!! For angles between 0 and 180 degress, "dy"
must be positive. Please modify the "Ramp Angle" or "dy". ','Invalid Analysis')
        return;

    end

end

end

%
% "Spring Energy" from stored power.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

%get PE data, stored spring
pe_k = str2double(get(handles.pe_k,'String'));
pe_xf = str2double(get(handles.pe_xf,'String'));
pe_xi = str2double(get(handles.pe_xi,'String'));
pe_loss = str2double(get(handles.pe_loss,'String'));
pe_mass = str2double(get(handles.pe_mass,'String'));

%set pe_loss as percentage
pe_loss = pe_loss / 100;

PE = 0;
%calc PE
if (get(handles.pe_check,'Value')) == 1

    PE = ( (0.5 * pe_k * pe_xf ^ 2) - (0.5 * pe_k * pe_xi ^ 2) )*(1-pe_loss);

end

angle = ke_angle;

%
% Combine Energies
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%get Launch Angle data
if (get(handles.la_check,'Value')) == 1
    angle = str2double(get(handles.la_angle,'String'));
end

%set launch magnitude and angle
energy = (KE + PE);

%inital velocity check
%only check if PE is turned off
if energy < 0
    errordlg('The total applied energy cannot overcome the ramp','Invalid Analysis')
    return;
end

%
%choose mass based on Ramp \ Spring Launch
%
% if both, the velocity is calculated from the vehicle mass
%if just spring, then use projectile mass,
%also use projectile mass for drag

if (get(handles.pe_check,'Value')) == 1 & (get(handles.ke_check,'Value')) == 1

    %use ramp mass to calc launch vel
    mass = ke_mass;

```

```

% angle determines the sign, enforce vlaunch as positive
if energy >= 0
    vlaunch = sqrt( 2 * energy / mass );
else
    errordlg('NEGATIVE KE VALUE','Invalid Analysis')
    return;
end

%but mod mass to account for projectile
mass = pe_mass;

elseif (get(handles.pe_check,'Value')) == 1
    mass = pe_mass;
    vlaunch = sqrt( 2 * energy / mass );
else
    mass = ke_mass;

    % angle determines the sign, enforce vlaunch as positive
    if KE >= 0
        vlaunch = sqrt( 2 * energy / mass );
    else
        errordlg('NEGATIVE KE VALUE','Invalid Analysis')
        return;
    end

end

%output vlaunch info to screen

fprintf("\n Launch Velocity = %g %s \n", vlaunch, unit_velocity)
fprintf(' at %g degrees CCW from the x-axis\n\n', angle)

%
% Launch
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%set initial conditions
angle_radian = angle * pi / 180;

vx(2) = vlaunch * cos(angle_radian);
vy(2) = vlaunch * sin(angle_radian);

ke_angle_radian = ke_angle * pi / 180;

if (get(handles.ke_check,'Value')) == 1
    x(1) = -(ke_dy / tan(ke_angle_radian));
    y(1) = ke_height;
end

x(2) = 0;
y(2) = ke_height + ke_dy;

```

```

%draw target
if (get(handles.td_check,'Value')) == 1

    %build target poly
    x_tar(1) = td_xmin;
    y_tar(1) = td_ymin;

    x_tar(2) = td_xmin;
    y_tar(2) = td_ymax;

    x_tar(3) = td_xmax;
    y_tar(3) = td_ymax;

    x_tar(4) = td_xmax;
    y_tar(4) = td_ymin;

    x_tar(5) = td_xmin;
    y_tar(5) = td_ymin;

    plot(x_tar, y_tar, 'r-')

    x_tar(5) = [];
    y_tar(5) = [];

end

%
% Particle Motion, No Drag
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%calc motion

index1 = 2;

y_max = 0;
hangtime = 0;
hit = 0;

while y(index1) >= 0 & hit == 0

    index1 = index1 + 1;
    hangtime = hangtime + dt;

    vx(index1) = vx(index1 - 1);
    vy(index1) = vy(index1 - 1) + gravity * dt;

    x(index1) = x(index1 - 1) + vx(index1 - 1) * dt;
    y(index1) = y(index1 - 1) + vy(index1 - 1) * dt + 0.5 * gravity * dt^2;

```

```

%check for hit target
if (get(handles.td_check,'Value')) == 1
    hit = inpolygon( x(index1), y(index1), x_tar, y_tar);
end

%if max, save data
if y(index1) > y_max

    y_max = y(index1);
    x_max = x(index1);
    time_max = hangtime;
end
end

%
% Drag
%
%
%
%get drag info and drag_type

%density_air = .002328;

% reset velocity
vx_drag(2) = vlaunch * cos(angle_radian);
vy_drag(2) = vlaunch * sin(angle_radian);

if (get(handles.drag_simple,'Value')) == 1
    drag_simple_loss = str2double(get(handles.drag_loss,'String'));

%
% Drag, simple
%
%
% initial data
x_drag(1) = x(1);
y_drag(1) = y(1);
x_drag(2) = x(2);
y_drag(2) = y(2);

%set initial conditions
angle_radian = angle * pi / 180;

vx_drag(2) = vlaunch * cos(angle_radian) * (100 - drag_simple_loss)/100;
vy_drag(2) = vlaunch * sin(angle_radian) * (100 - drag_simple_loss)/100;

%calc motion
index1 = 2;

y_drag_max = 0;
hangtime_drag = dt;

```

```

hit = 0;

while y(index1) >= 0 & hit == 0

    index1 = index1 + 1;
    %calc hangtime
    hangtime_drag = hangtime_drag + dt;

    %change in vel from gravity
    vx_drag(index1) = vx_drag(index1 - 1);
    vy_drag(index1) = vy_drag(index1 - 1) + gravity * dt;

    %new position
    x_drag(index1) = x_drag(index1 - 1) + vx_drag(index1-1) * dt;
    y_drag(index1) = y_drag(index1 - 1) + vy_drag(index1-1) * dt + 0.5*gravity*dt^2;

    %check for hit target
    if (get(handles.td_check,'Value')) == 1
        hit = inpolygon( x_drag(index1), y_drag(index1), x_tar, y_tar);
    end

    %if max, save data
    if y_drag(index1) > y_drag_max

        y_drag_max = y_drag(index1);
        x_drag_max = x_drag(index1);
        time_drag_max = hangtime_drag;
    end

end

elseif (get(handles.drag_complex,'Value')) == 1

    %
    % Drag, complex
    %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    drag_k = str2double(get(handles.drag_k,'String'));
    drag_density = str2double(get(handles.drag_density,'String'));

    % initial data
    x_drag(1) = x(1);
    y_drag(1) = y(1);
    x_drag(2) = x(2);
    y_drag(2) = y(2);

    %set initial conditions
    angle_radian = angle * pi / 180;

    vx_drag(2) = vlaunch * cos(angle_radian);

```



```

vy_drag(2) = vlaunch * sin(angle_radian);

%impulse
index1 = 2;

y_drag_max = 0;
hangtime_drag = 0;
hit = 0;

lamda = drag_k / mass;

while y_drag(index1) >= 0 & hit == 0

    index1 = index1 + 1;

    %determine x and y position
    x_drag(index1) = x_drag(2) + vx_drag(2)*(1-exp(-lamda*hangtime_drag))/lamda;
    y_drag(index1) = y_drag(2) + (-gravity / lamda^2 + vy_drag(2)/lamda)*(1-exp(-
lamda*hangtime_drag)) + (gravity/lamda)*hangtime_drag;

    %bump time
    hangtime_drag = hangtime_drag + dt;

    %calc drag velocity
    vx_drag(index1) = (x_drag(index1) - x_drag(index1-1))/dt;
    vy_drag(index1) = (y_drag(index1) - y_drag(index1-1))/dt;

    %check for hit target
    if (get(handles.td_check,'Value')) == 1
        hit = inpolygon( x_drag(index1), y_drag(index1), x_tar, y_tar);
    end

    %if max, save data
    if y_drag(index1) > y_drag_max

        y_drag_max = y_drag(index1);
        x_drag_max = x_drag(index1);
        time_drag_max = hangtime_drag;
    end

end

end

%get draw results info
switch get(handles.ShowResultsMenu,'Checked')
case 'on'
    tog_results = 1;

```

```

        case 'off'
            tog_results = 0;
        end

switch get(handles.ShowSimpleMenu,'Checked')
    case 'on'
        tog_simple = 1;
    case 'off'
        tog_simple = 0;
    end

switch get(handles.ShowDragMenu,'Checked')
    case 'on'
        tog_drag = 1;
    case 'off'
        tog_drag = 0;
    end

%
%
%plot
if (get(handles.figure_check,'Value')) == 1
    figure(1)
    title('Projectile Motion Results')

else
    axes(handles.axes1);

end

hold on;

xlabel(['Distance ( ',unit_position,' )'])
ylabel(['Height ( ',unit_position,' )'])

%draw ramp
xx(1) = x(1);
xx(2) = x(2);
xx(3) = 0;
xx(4) = x(1);
xx(5) = x(1);

yy(1) = y(1);
yy(2) = y(2);

```

```

yy(3) = 0;
yy(4) = 0;
yy(5) = y(1);

%plot ramp
fill(xx,yy,'r')
plot(xx,yy,'k-')

%draw landing info and plot path
switch get(handles.ShowSimpleMenu,'Checked')
case 'on'
    plot(x,y,'b*-')

    if tog_results == 1

        [a b] = size(x);

        text(x(b) , 0,[' ',num2str(x(b),'%9.3f'),...
            ' ',unit_position,' at ',num2str(hangtime),' seconds '],...
            'BackgroundColor',[.1 .7 .9],...
            'EdgeColor','k',...
            'HorizontalAlignment','center',...
            'VerticalAlignment','bottom')

    end

case 'off'
    %nadda
end

%plot drag
switch get(handles.ShowDragMenu,'Checked')
case 'on'
    if (get(handles.drag_complex,'Value')) == 1 | (get(handles.drag_simple,'Value')) == 1

        plot(x_drag, y_drag,'g*-')

        %draw results
        if tog_results == 1

            [a b] = size(x_drag);

            text(x_drag(b), -x_drag(b)/30,[' ',num2str(x_drag(b),'%9.3f'),' ',unit_position,' ',...
                ' at ',num2str(hangtime_drag,'%9.3f'),' seconds '],...
                'BackgroundColor',[.1 .9 .5],...
                'EdgeColor','k',...

```

```

        'HorizontalAlignment','center',...
        'VerticalAlignment','top')
    end

end

case 'off'
    %nadda
end

%set axis to display results
axis auto

%draw results

axis auto;
axis equal
user_axis = axis;

x_length = abs(user_axis(1) - user_axis(2)).*0.15;
y_length = abs(user_axis(3) - user_axis(4)).*0.15;

%if draw results and this path
if tog_results == 1 & tog_simple == 1

    text(x_max - x_length*2.0, y_max + y_length*1.5, 'No Drag Ymax Data', 'HorizontalAlignment','center')
    text(x_max - x_length*2.0, y_max + y_length*1.0, [' X position = ', num2str(x_max, '%9.3f'), '
', unit_position, ' '], ...
        'BackgroundColor',[.8 .8 .8],...
        'EdgeColor','k',...
        'HorizontalAlignment','center',...
        'VerticalAlignment','top')
    text(x_max - x_length*2.0, y_max + y_length*0.5, [' Y position = ', num2str(y_max, '%9.3f'), '
', unit_position, ' '], ...
        'BackgroundColor',[.8 .8 .8],...
        'EdgeColor','k',...
        'HorizontalAlignment','center',...
        'VerticalAlignment','top')
    text(x_max - x_length*2, y_max + y_length*0, [' Time = ', num2str(time_max, 4), ' seconds '], ...
        'BackgroundColor',[.8 .8 .8],...
        'EdgeColor','k',...
        'HorizontalAlignment','center',...
        'VerticalAlignment','top')

plot(x_max, y_max, 'ro',...
    'MarkerEdgeColor','k',...
    'MarkerFaceColor','r',...
    'MarkerSize',10)

```

```

end

%check for drag values
if (get(handles.drag_complex,'Value')) == 1 | (get(handles.drag_simple,'Value')) == 1
    %already set value
    tog_drag = tog_drag;
else
    tog_drag = 0;
end

%if draw results and this path
if tog_results == 1 & tog_drag == 1

    text(x_drag_max - x_length*1.5, y_drag_max - y_length*1,'Drag Path Ymax
Data','HorizontalAlignment','center')
    text(x_drag_max - x_length*1.5, y_drag_max - y_length*1.6,[' X position =
',num2str(x_drag_max,'%9.3f'),' ',unit_position,' '],...
        'BackgroundColor',[.8 .8 .8],...
        'EdgeColor','k',...
        'HorizontalAlignment','center',...
        'VerticalAlignment','bottom')
    text(x_drag_max - x_length*1.5, y_drag_max - y_length*2.1,[' Y position =
',num2str(y_drag_max,'%9.3f'),' ',unit_position,' '],...
        'BackgroundColor',[.8 .8 .8],...
        'EdgeColor','k',...
        'HorizontalAlignment','center',...
        'VerticalAlignment','bottom')
    text(x_drag_max - x_length*1.5, y_drag_max - y_length*2.6,[' Time = ',num2str(time_drag_max,4), '
seconds '],...
        'BackgroundColor',[.8 .8 .8],...
        'EdgeColor','k',...
        'HorizontalAlignment','center',...
        'VerticalAlignment','bottom')

    plot(x_drag_max, y_drag_max, 'ro',...
        'MarkerEdgeColor','k',...
        'MarkerFaceColor','r',...
        'MarkerSize',10)

end

%get/modify axis information
mod_axis = axis;

```

```

mod_axis(1) = (mod_axis(1))*0.95;
mod_axis(2) = (mod_axis(2))*1.05;
mod_axis(3) = 0; % always set to zero
mod_axis(4) = (mod_axis(4))*1.3;

```

```

axis([mod_axis])

```

```

hold off;

```

```

%if output data to command window
if (get(handles.data_check,'Value')) == 1

```

```

figure(2)
clf reset;
hold on;

```

```

subplot(2,1,1)

```

```

hold on;
title('Velocity vs. X position')
xlabel(['Distance (',unit_position,')'])
ylabel(['Velocity (',unit_velocity,')'])

```

```

%remove swing data point

```

```

vx(1) = [];
vy(1) = [];
y(1) = [];
x(1) = [];

```

```

vmag = sqrt(vx.^2 + vy.^2);

```

```

plot(x, vmag, 'b')

```

```

%if drag exists, plot
if length(vx_drag) > 3

```

```

%format data to adjust for the ramp

```

```

vx_drag(1) = [];
vx_drag(1) = [];
%vx_drag(1) = [];

```

```

vy_drag(1) = [];
vy_drag(1) = [];
%vy_drag(1) = [];

```

```

%set initial velocity data for plot
vx_drag(1) = vx(1);
vy_drag(1) = vy(1);
%
x_drag(1) = [];
x_drag(1) = [];
%vx_drag(1) = [];

y_drag(1) = [];
y_drag(1) = [];
%y_drag(1) = [];

vmag_drag = sqrt(vx_drag.^2 + vy_drag.^2);

plot(x_drag, vmag_drag, 'g')

legend('No Drag','With Drag',4)
end

%calc angle of velocity
v_angle = atan2(vy, vx).* 180 ./ pi;
%v_angle(1) = [];

subplot(2,1,2)
hold on;

title('Theta vs. X position')
xlabel(['Distance (',unit_position,' )'])
ylabel(['Theta ( degrees )'])

plot(x, v_angle, 'b')

%if drag exists, plot
if length(vx_drag) > 3

    %calc angle of velocity
    v_angle_drag = atan2(vy_drag, vx_drag).* 180 ./ pi;

    plot(x_drag, v_angle_drag, 'g')

    legend('No Drag','With Drag',1)

end

%output data as a table
clc

time =0;

fid = fopen('EFD_RAMP_output.txt','wb');

```

```

fprintf(fid,'\n-----');
fprintf(fid,'\n----- Simple Model Data -----');
fprintf(fid,'\n\n   time       X           Y           Vx           Vy ');
fprintf(fid,'\n ');

%display units
fprintf(fid,'\n [ seconds ]      [ %s ]      [ %s ]',unit_position,unit_velocity);
fprintf(fid,'\n-----');
for i = 1:length(x)
    fprintf(fid,'\n %7.3f \t%9.3f \t%9.3f \t%9.3f \t%9.3f', time, x(i),y(i),vx(i),vy(i));

    time = time + dt;
end

fclose('all');

%print text file to HTML
web(['file:' which('EFD_RAMP_output.txt')],'-browser')

%if drag data
if length(vx_drag) > 3

    time = 0;

    fid = fopen('EFD_RAMP_output_DRAG.txt','wb');

    fprintf(fid,'\n-----');
    fprintf(fid,'\n----- Drag Model Data -----');
    fprintf(fid,'\n\n   time       X           Y           Vx           Vy ');
    fprintf(fid,'\n ');

    %display units
    fprintf(fid,'\n [ seconds ]      [ %s ]      [ %s ]',unit_position,unit_velocity);
    fprintf(fid,'\n-----');

    for i = 1:length(x_drag)
        fprintf(fid,'\n %7.3f \t%9.3f \t%9.3f \t%9.3f \t%9.3f', time,
x_drag(i),y_drag(i),vx_drag(i),vy_drag(i));

        time = time + dt;
    end

    fclose('all');

    %print text file to HTML
    web(['file:' which('EFD_RAMP_output_DRAG.txt')],'-browser')

end
end

% -----

```



```

function FileMenu_Callback(hObject, eventdata, handles)
% hObject   handle to FileMenu (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

% -----
function OpenMenuItem_Callback(hObject, eventdata, handles)
% hObject   handle to OpenMenuItem (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
file = uigetfile('*.fig');
if ~isequal(file, 0)
    open(file);
end

% -----
function PrintMenuItem_Callback(hObject, eventdata, handles)
% hObject   handle to PrintMenuItem (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
printdlg(handles.figure1)

% -----
function CloseMenuItem_Callback(hObject, eventdata, handles)
% hObject   handle to CloseMenuItem (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
selection = questdlg(['Close ' get(handles.figure1,'Name') '?'],...
    ['Close ' get(handles.figure1,'Name') '...'],...
    'Yes','No','Yes');
if strcmp(selection,'No')
    return;
end

delete(handles.figure1)

% -----
function ViewMenu_Callback(hObject, eventdata, handles)
% hObject   handle to ViewMenu (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

% -----
function DisplayMenu_Callback(hObject, eventdata, handles)
% hObject   handle to DisplayMenu (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

% -----
function ShowGridMenu_Callback(hObject, eventdata, handles)
% hObject   handle to ShowGridMenu (see GCBO)

```

```
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
```

```
% -----
function ErrorCheckMenu_Callback(hObject, eventdata, handles)
% hObject handle to ErrorCheckMenu (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
```

```
% -----
function HelpMenu_Callback(hObject, eventdata, handles)
% hObject handle to HelpMenu (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
```

```
% -----
function HelpGuideMenu_Callback(hObject, eventdata, handles)
% hObject handle to HelpGuideMenu (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
```

```
web(['file:' which('help_home.html')],'-browser')
```

```
% -----
function HelpAboutMenu_Callback(hObject, eventdata, handles)
% hObject handle to HelpAboutMenu (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
```

```
msgbox('Projectile Motion Analysis for Ramp and Spring Launching,  
, 'About this program', 'help')
```

written by Jon Huber'

```
% --- Executes on button press in ke_check.
function ke_check_Callback(hObject, eventdata, handles)
% hObject handle to ke_check (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
```

```
% Hint: get(hObject,'Value') returns toggle state of ke_check
```

```
% --- Executes during object creation, after setting all properties.
function ke_vmag_CreateFcn(hObject, eventdata, handles)
% hObject handle to ke_vmag (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
```

```

else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function ke_vmag_Callback(hObject, eventdata, handles)
% hObject    handle to ke_vmag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ke_vmag as text
%        str2double(get(hObject,'String')) returns contents of ke_vmag as a double

% --- Executes during object creation, after setting all properties.
function ke_angle_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ke_angle (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function ke_angle_Callback(hObject, eventdata, handles)
% hObject    handle to ke_angle (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ke_angle as text
%        str2double(get(hObject,'String')) returns contents of ke_angle as a double

% --- Executes during object creation, after setting all properties.
function ke_height_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ke_height (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

```

```

function ke_height_Callback(hObject, eventdata, handles)
% hObject    handle to ke_height (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ke_height as text
%        str2double(get(hObject,'String')) returns contents of ke_height as a double

% --- Executes on button press in pe_check.
function pe_check_Callback(hObject, eventdata, handles)
% hObject    handle to pe_check (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of pe_check

% --- Executes during object creation, after setting all properties.
function pe_k_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pe_k (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function pe_k_Callback(hObject, eventdata, handles)
% hObject    handle to pe_k (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of pe_k as text
%        str2double(get(hObject,'String')) returns contents of pe_k as a double

% --- Executes during object creation, after setting all properties.
function pe_xf_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pe_xf (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.

```

```

%    See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function pe_xf_Callback(hObject, eventdata, handles)
% hObject    handle to pe_xf (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of pe_xf as text
%    str2double(get(hObject,'String')) returns contents of pe_xf as a double

% --- Executes during object creation, after setting all properties.
function drag_density_CreateFcn(hObject, eventdata, handles)
% hObject    handle to drag_density (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%    See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function drag_density_Callback(hObject, eventdata, handles)
% hObject    handle to drag_density (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of drag_density as text
%    str2double(get(hObject,'String')) returns contents of drag_density as a double

% --- Executes during object creation, after setting all properties.
function la_angle_CreateFcn(hObject, eventdata, handles)
% hObject    handle to la_angle (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%    See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');

```

```

else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

```

```

function la_angle_Callback(hObject, eventdata, handles)
% hObject    handle to la_angle (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of la_angle as text
%        str2double(get(hObject,'String')) returns contents of la_angle as a double

```

```

% --- Executes during object creation, after setting all properties.
function drag_k_CreateFcn(hObject, eventdata, handles)
% hObject    handle to drag_k (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

```

```

function drag_k_Callback(hObject, eventdata, handles)
% hObject    handle to drag_k (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of drag_k as text
%        str2double(get(hObject,'String')) returns contents of drag_k as a double

```

```

% --- Executes on button press in la_check.
function la_check_Callback(hObject, eventdata, handles)
% hObject    handle to la_check (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

% Hint: get(hObject,'Value') returns toggle state of la_check

```

```

% --- Executes on button press in drag_simple.
function drag_simple_Callback(hObject, eventdata, handles)

```

```

% hObject    handle to drag_simple (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of drag_simple
set(handles.drag_no, 'Value', 0);
set(handles.drag_simple, 'Value', 1);
set(handles.drag_complex, 'Value', 0);

% --- Executes on button press in drag_complex.
function drag_complex_Callback(hObject, eventdata, handles)
% hObject    handle to drag_complex (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of drag_complex

set(handles.drag_no, 'Value', 0);
set(handles.drag_simple, 'Value', 0);
set(handles.drag_complex, 'Value', 1);

% --- Executes during object creation, after setting all properties.
function ti_dt_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ti_dt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function ti_dt_Callback(hObject, eventdata, handles)
% hObject    handle to ti_dt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ti_dt as text
%       str2double(get(hObject,'String')) returns contents of ti_dt as a double

% --- Executes during object creation, after setting all properties.
function ke_loss_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ke_loss (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

```

```

% Hint: edit controls usually have a white background on Windows.
%     See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

```

```

function ke_loss_Callback(hObject, eventdata, handles)
% hObject    handle to ke_loss (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ke_loss as text
%        str2double(get(hObject,'String')) returns contents of ke_loss as a double

```

```

% --- Executes during object creation, after setting all properties.
function pe_loss_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pe_loss (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

```

```

% Hint: edit controls usually have a white background on Windows.
%     See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

```

```

function pe_loss_Callback(hObject, eventdata, handles)
% hObject    handle to pe_loss (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of pe_loss as text
%        str2double(get(hObject,'String')) returns contents of pe_loss as a double

```

```

% --- Executes on button press in drag_no.
function drag_no_Callback(hObject, eventdata, handles)
% hObject    handle to drag_no (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

% Hint: get(hObject,'Value') returns toggle state of drag_no

```

```

set(handles.drag_no, 'Value', 1);

```



```

set(handles.drag_simple, 'Value', 0);
set(handles.drag_complex, 'Value', 0);

% --- Executes during object creation, after setting all properties.
function drag_loss_CreateFcn(hObject, eventdata, handles)
% hObject    handle to drag_loss (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function drag_loss_Callback(hObject, eventdata, handles)
% hObject    handle to drag_loss (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of drag_loss as text
%       str2double(get(hObject,'String')) returns contents of drag_loss as a double

% --- Executes during object creation, after setting all properties.
function ke_mass_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ke_mass (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function ke_mass_Callback(hObject, eventdata, handles)
% hObject    handle to ke_mass (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ke_mass as text
%       str2double(get(hObject,'String')) returns contents of ke_mass as a double

```

```

% -----
function UnitsMenu_Callback(hObject, eventdata, handles)
% hObject    handle to UnitsMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% -----
function UnitsEnglish_Callback(hObject, eventdata, handles)
% hObject    handle to UnitsEnglish (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

set(handles.UnitsEnglish, 'Checked', 'on');
set(handles.UnitsMetric, 'Checked', 'off');

%pop-up figure with units menu
figure(2);
clf reset;
axis([0 100 0 100])

title('English System of Units')

text(5,96,'Time Increment')
text(5,93,'-----')
text(5,90,'dt == ( sec )')

text(5,81,'Calculate Ramp Energy')
text(5,78,'-----')
text(5,75,'Vmag == ( ft / sec )')
text(5,70,'angle == ( degrees )')
text(5,65,'height == ( ft )')
text(5,60,'dy == ( ft )')
text(5,55,'mass of vechicle == ( slugs )')
text(5,50,'loss == ( value of 0 thru 1 )')

text(5,41,'Calculate Launch Energy')
text(5,38,'-----')
text(5,35,'k == ( pounds / ft )')
text(5,30,'x == ( ft )')
text(5,25,'mass of projectile == ( slugs )')
text(5,20,'dy == ( ft )')
text(5,15,'mass of vechicle == ( slugs )')
text(5,10,'loss == ( value of 0 thru 1 )')

text(55,96,'Aim for Target')
text(55,93,'-----')
text(55,90,'Xmin, Xmax == ( ft )')
text(55,85,'Ymin, Ymax == ( ft )')

text(55,76,'Specific Launch Angle')
text(55,73,'-----')
text(55,70,'angle == ( degrees )')

```

```

text(55,61,'Drag Menu')
text(55,58,'-----')
text(55,55,'loss == ( value of 0 thru 1 )')
text(55,50,'k == ( lb * s / ft )')
text(55,45,'density of air == ( slug / ft^3 )')

% -----
function UnitsMetric_Callback(hObject, eventdata, handles)
% hObject    handle to UnitsMetric (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

set(handles.UnitsEnglish, 'Checked', 'off');
set(handles.UnitsMetric, 'Checked', 'on');

%pop-up figure with units menu
figure(2);
clf reset;
axis([0 100 0 100])

title('Metric System of Units')

text(5,96,'Time Increment')
text(5,93,'-----')
text(5,90,'dt == ( sec )')

text(5,81,'Calculate Ramp Energy')
text(5,78,'-----')
text(5,75,'Vmag == ( m / sec )')
text(5,70,'angle == ( degrees )')
text(5,65,'height == ( m )')
text(5,60,'dy == ( m )')
text(5,55,'mass of vechicle == ( kg )')
text(5,50,'loss == ( value of 0 thru 100 )')

text(5,41,'Calculate Launch Energy')
text(5,38,'-----')
text(5,35,'k == ( N / m )')
text(5,30,'x == ( m )')
text(5,25,'mass of projectile == ( kg )')
text(5,20,'dy == ( m )')
text(5,15,'mass of vechicle == ( kg )')
text(5,10,'loss == ( value of 0 thru 100 )')

text(55,96,'Aim for Target')
text(55,93,'-----')
text(55,90,'Xmin, Xmax == ( m )')
text(55,85,'Ymin, Ymax == ( m )')

text(55,76,'Specific Launch Angle')
text(55,73,'-----')

```

```

text(55,70,'angle == ( degrees )')

text(55,61,'Drag Menu')
text(55,58,'-----')
text(55,55,'loss == ( value of 0 thru 100 )')
text(55,50,'k == ( N * s / m )')
text(55,45,'density of air == ( kg / m^3 )')

% --- Executes during object creation, after setting all properties.
function ke_dy_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ke_dy (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function ke_dy_Callback(hObject, eventdata, handles)
% hObject    handle to ke_dy (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ke_dy as text
%       str2double(get(hObject,'String')) returns contents of ke_dy as a double

% -----
function UnitsRampMenu_Callback(hObject, eventdata, handles)
% hObject    handle to UnitsRampMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% -----
function UnitsLaunchMenu_Callback(hObject, eventdata, handles)
% hObject    handle to UnitsLaunchMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function pe_mass_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pe_mass (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

```

```

% Hint: edit controls usually have a white background on Windows.
%     See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function pe_mass_Callback(hObject, eventdata, handles)
% hObject    handle to pe_mass (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of pe_mass as text
%        str2double(get(hObject,'String')) returns contents of pe_mass as a double

% -----
function ShowResultsMenu_Callback(hObject, eventdata, handles)
% hObject    handle to ShowResultsMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

switch get(handles.ShowResultsMenu,'Checked')
    case 'on'
        set(handles.ShowResultsMenu, 'Checked', 'off')
    case 'off'
        set(handles.ShowResultsMenu, 'Checked', 'on')
end

% -----
function ShowSimpleMenu_Callback(hObject, eventdata, handles)
% hObject    handle to ShowSimpleMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

switch get(handles.ShowSimpleMenu,'Checked')
    case 'on'
        set(handles.ShowSimpleMenu, 'Checked', 'off')
    case 'off'
        set(handles.ShowSimpleMenu, 'Checked', 'on')
end

% -----
function ShowDragMenu_Callback(hObject, eventdata, handles)
% hObject    handle to ShowDragMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

switch get(handles.ShowDragMenu,'Checked')
    case 'on'
        set(handles.ShowDragMenu, 'Checked', 'off')

```

```

        case 'off'
            set(handles.ShowDragMenu, 'Checked', 'on')
        end

% --- Executes on button press in figure_check.
function figure_check_Callback(hObject, eventdata, handles)
% hObject    handle to figure_check (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of figure_check

% --- Executes on button press in td_check.
function td_check_Callback(hObject, eventdata, handles)
% hObject    handle to td_check (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of td_check

% --- Executes during object creation, after setting all properties.
function td_xmin_CreateFcn(hObject, eventdata, handles)
% hObject    handle to td_xmin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function td_xmin_Callback(hObject, eventdata, handles)
% hObject    handle to td_xmin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of td_xmin as text
%       str2double(get(hObject,'String')) returns contents of td_xmin as a double

% --- Executes during object creation, after setting all properties.
function td_xmax_CreateFcn(hObject, eventdata, handles)
% hObject    handle to td_xmax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFns called

```

```

% Hint: edit controls usually have a white background on Windows.
%     See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function td_xmax_Callback(hObject, eventdata, handles)
% hObject    handle to td_xmax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of td_xmax as text
%        str2double(get(hObject,'String')) returns contents of td_xmax as a double

% --- Executes during object creation, after setting all properties.
function td_ymin_CreateFcn(hObject, eventdata, handles)
% hObject    handle to td_ymin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%     See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function td_ymin_Callback(hObject, eventdata, handles)
% hObject    handle to td_ymin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of td_ymin as text
%        str2double(get(hObject,'String')) returns contents of td_ymin as a double

% --- Executes during object creation, after setting all properties.
function td_ymax_CreateFcn(hObject, eventdata, handles)
% hObject    handle to td_ymax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%     See ISPC and COMPUTER.

```

```

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function td_ymax_Callback(hObject, eventdata, handles)
% hObject    handle to td_ymax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of td_ymax as text
%        str2double(get(hObject,'String')) returns contents of td_ymax as a double

% --- Executes during object creation, after setting all properties.
function pe_xi_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pe_xi (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function pe_xi_Callback(hObject, eventdata, handles)
% hObject    handle to pe_xi (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of pe_xi as text
%        str2double(get(hObject,'String')) returns contents of pe_xi as a double

% --- Executes on button press in data_check.
function data_check_Callback(hObject, eventdata, handles)
% hObject    handle to data_check (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of data_check

```


Appendix B.3

Swing Energy Code

EFD_BALLDROP

```
function varargout = efd_balldrop(varargin)
% EFD_BALLDROP M-file for efd_balldrop.fig
%   EFD_BALLDROP, by itself, creates a new EFD_BALLDROP or raises the existing
%   singleton*.
%
%   H = EFD_BALLDROP returns the handle to a new EFD_BALLDROP or the handle to
%   the existing singleton*.
%
%   EFD_BALLDROP('CALLBACK',hObject,eventData,handles,...) calls the local
%   function named CALLBACK in EFD_BALLDROP.M with the given input arguments.
%
%   EFD_BALLDROP('Property','Value',...) creates a new EFD_BALLDROP or raises the
%   existing singleton*. Starting from the left, property value pairs are
%   applied to the GUI before efd_balldrop_OpeningFunction gets called. An
%   unrecognized property name or invalid value makes property application
%   stop. All inputs are passed to efd_balldrop_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
%   instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help efd_balldrop

% Last Modified by GUIDE v2.5 01-Jun-2003 17:35:21

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',    mfilename, ...
    'gui_Singleton', gui_Singleton, ...
    'gui_OpeningFcn', @efd_balldrop_OpeningFcn, ...
    'gui_OutputFcn', @efd_balldrop_OutputFcn, ...
    'gui_LayoutFcn', [] , ...
    'gui_Callback', []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```

% --- Executes just before efd_balldrop is made visible.
function efd_balldrop_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to efd_balldrop (see VARARGIN)

% Choose default command line output for efd_balldrop
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% This sets up the initial plot - only do when we are invisible
% so window can get raised using efd_balldrop.
if strcmp(get(hObject,'Visible'),'off')
    surf(peaks);
end

set(handles.se_check, 'Value', 1);
set(handles.se_same, 'Value', 1);
set(handles.drag_no, 'Value', 1);
set(handles.UnitsEnglish, 'Checked', 'off');
set(handles.UnitsMetric, 'Checked', 'on');

% UIWAIT makes efd_balldrop wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = efd_balldrop_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%get output info
if (get(handles.figure_check,'Value')) == 1
    figure(1)

```

```

        clf reset;
    else
        axes(handles.axes1);
        cla;
    end

    %set view
    view(2)

    clc

    %%
    %determine units
    switch get(handles.UnitsEnglish,'Checked')
    case 'on'
        gravity = -32.2;
        unit_position = 'feet';
    case 'off'
        gravity = -9.8102;
        unit_position = 'meters';
    end

    %get Properties data
    dt = str2double(get(handles.prop_dt,'String'));
    prop_vt = str2double(get(handles.prop_vt,'String'));
    prop_mass = str2double(get(handles.prop_mass,'String'));

    %get Swing Energy data
    se_x = str2double(get(handles.se_x,'String'));
    se_y = str2double(get(handles.se_y,'String'));
    se_r = str2double(get(handles.se_r,'String'));
    se_start_angle = str2double(get(handles.se_start_angle,'String'));
    se_release_angle = str2double(get(handles.se_release_angle,'String'));
    se_loss = str2double(get(handles.se_loss,'String'));
    se_specify = str2double(get(handles.se_specify_angle,'String'));
    se_offset = str2double(get(handles.se_offset_angle,'String'));

    particle_mass = str2double(get(handles.particle_mass,'String'));
    mass = particle_mass;
    %get Target data
    td_xmin = str2double(get(handles.td_xmin,'String'));
    td_xmax = str2double(get(handles.td_xmax,'String'));
    td_ymin = str2double(get(handles.td_ymin,'String'));
    td_ymax = str2double(get(handles.td_ymax,'String'));

    vx_drag = 0;
    %ready to plot output

    %get release angle information

```

```

%
%
%

%
%
%
%
KE = 0;

%set angle based on particle release
if (get(handles.se_same,'Value')) == 1
    angle = se_release_angle + 90;
elseif (get(handles.se_specify,'Value')) == 1
    angle = 90 + se_specify;
else
    angle = se_release_angle + 90 + se_offset;
end

angle_radian = angle * pi / 180;

%draw full rotation , then swing path

if (get(handles.se_check,'Value')) == 1

    %full rotation

    count = 0;

    for i = 0:5:365

        count = count + 1;
        ang = i * 3.14 / 180;
        x_rot(count) = se_r * cos(ang);
        y_rot(count) = se_r * sin(ang);

    end

    x_rot = x_rot + se_x;
    y_rot = y_rot + se_y;

    plot(x_rot, y_rot, 'r-')

    hold on;

    plot(se_x, se_y, 'r*')

    %draw swing path
    count = 0;

    for i = se_start_angle:5:se_release_angle

```

```

        count = count + 1;
        ang = i * 3.14 / 180;
        x_swing(count) = se_r * cos(ang);
        y_swing(count) = se_r * sin(ang);

    end

    %shift circle to new X Y
    x_swing = x_swing + se_x;
    y_swing = y_swing + se_y;

    plot(x_swing, y_swing, 'b-')

    %calc release kinetic energy
    %    calc dy

    dy = se_r * sin(se_release_angle * pi / 180) - ...
        se_r * sin(se_start_angle * pi / 180);

    KE = (0.5 * prop_mass * prop_vt ^ 2 + gravity * prop_mass * dy) * (100-se_loss)/100;

    if KE > 0
        vlaunch = sqrt( 2 * KE / particle_mass );
    else
        errordlg('The initial velocity cannot reach the release point','Invalid Analysis')
        return;
    end

    %set initial conditions

    vx(1) = vlaunch * cos(angle_radian);
    vy(1) = vlaunch * sin(angle_radian);
    %sets final x,y position
    x(1) = se_r * cos(se_release_angle * pi / 180) + se_x;
    y(1) = se_r * sin(se_release_angle * pi / 180) + se_y;

end
hold on;
axis equal

%
%
%draw target
if (get(handles.td_check,'Value')) == 1

```

```

%build target poly
x_tar(1) = td_xmin;
y_tar(1) = td_ymin;

x_tar(2) = td_xmin;
y_tar(2) = td_ymax;

x_tar(3) = td_xmax;
y_tar(3) = td_ymax;

x_tar(4) = td_xmax;
y_tar(4) = td_ymin;

x_tar(5) = td_xmin;
y_tar(5) = td_ymin;

plot(x_tar, y_tar, 'r-')

end

%release, calc motion

%calc motion
index1 = 1;
hangtime = 0;
y_max = 0;
hit = 0;

%
%      Motion, no drag
%
%%%%%%%%%%
while y(index1) >= 0 & hit == 0

    index1 = index1 + 1;
    %calc hangtime
    hangtime = hangtime + dt;

    %change in velocity from gravity
    vx(index1) = vx(index1 - 1);
    vy(index1) = vy(index1 - 1) + gravity * dt;

    %new position
    x(index1) = x(index1 - 1) + vx(index1-1) * dt;
    y(index1) = y(index1 - 1) + vy(index1-1) * dt + 0.5*gravity*dt^2;

    %check for hit target
    if (get(handles.td_check,'Value')) == 1
        hit = inpolygon( x(index1), y(index1), x_tar, y_tar);
    end
end

```

```

%if max, save data
if y(index1) > y_max

    y_max = y(index1);
    x_max = x(index1);
    time_max = hangtime;
end

end

%
%    Drag, Simple
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%get drag info and drag_type

% reset velocity
% vx_drag(2) = vlaunch * cos(angle_radian);
% vy_drag(2) = vlaunch * sin(angle_radian);

if (get(handles.drag_simple,'Value')) == 1
    drag_simple_loss = str2double(get(handles.drag_loss,'String'));

    %
    % Drag, simple
    %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % initial data
    x_drag(1) = x(1);
    y_drag(1) = y(1);
    %x_drag(2) = x(2);
    %y_drag(2) = y(2);

    %set initial conditions
    angle_radian = angle * pi / 180;

    vx_drag(1) = vlaunch * cos(angle_radian) * (100 - drag_simple_loss)/100;
    vy_drag(1) = vlaunch * sin(angle_radian) * (100 - drag_simple_loss)/100;

    %calc motion
    index1 = 1;

    y_drag_max = 0;
    hangtime_drag = 0;
    hit = 0;

    while y(index1) >= 0 & hit == 0

        index1 = index1 + 1;

```

```

%calc hangtime
hangtime_drag = hangtime_drag + dt;

%change in velocity from gravity
vx_drag(index1) = vx_drag(index1 - 1);
vy_drag(index1) = vy_drag(index1 - 1) + gravity * dt;

%new pos
x_drag(index1) = x_drag(index1 - 1) + vx_drag(index1-1) * dt;
y_drag(index1) = y_drag(index1 - 1) + vy_drag(index1-1) * dt + 0.5*gravity*dt^2;

%check for hit target
if (get(handles.td_check,'Value')) == 1
    hit = inpolygon( x_drag(index1), y_drag(index1), x_tar, y_tar);
end

%if max, save data
if y_drag(index1) > y_drag_max

    y_drag_max = y_drag(index1);
    x_drag_max = x_drag(index1);
    time_drag_max = hangtime_drag;
end

end

elseif (get(handles.drag_complex,'Value')) == 1

%
% Drag, complex
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
drag_k = str2double(get(handles.drag_k,'String'));
drag_density = str2double(get(handles.drag_density,'String'));

% initial data
x_drag(1) = x(1);
y_drag(1) = y(1);
x_drag(2) = x(2);
y_drag(2) = y(2);

%set initial conditions
angle_radian = (angle) * pi / 180;

%vx_drag(1) = vlaunch * cos(angle_radian);
%vy_drag(1) = vlaunch * sin(angle_radian);

vx_drag(1) = vx(1)
vy_drag(1) = vy(1)

```



```

%impulse
index1 = 1;

y_drag_max = 0;
hangtime_drag = 0;
hit = 0;

lamda = drag_k / mass;

while y_drag(index1) >= 0 & hit == 0

    index1 = index1 + 1;

    %determine x and y position
    x_drag(index1) = x_drag(1) + vx_drag(1)*(1-exp(-lamda*hangtime_drag))/lamda;
    y_drag(index1) = y_drag(1) + (-gravity / lamda^2 + vy_drag(1)/lamda)*(1-exp(-
lamda*hangtime_drag)) + (gravity/lamda)*hangtime_drag;

    %bump time
    hangtime_drag = hangtime_drag + dt;

    vx_drag(index1) = (x_drag(index1) - x_drag(index1-1))/dt;
    vy_drag(index1) = (y_drag(index1) - y_drag(index1-1))/dt;

    %check for hit target
    if (get(handles.td_check,'Value')) == 1
        hit = inpolygon( x_drag(index1), y_drag(index1), x_tar, y_tar);
    end

    %if max, save data
    if y_drag(index1) > y_drag_max

        y_drag_max = y_drag(index1);
        x_drag_max = x_drag(index1);
        time_drag_max = hangtime_drag;
    end

end

end

%get draw results info
switch get(handles.ShowResultsMenu,'Checked')
case 'on'
    tog_results = 1;

case 'off'
    tog_results = 0;

```

```

end

switch get(handles.ShowSimpleMenu,'Checked')
    case 'on'
        tog_simple = 1;
    case 'off'
        tog_simple = 0;
end

switch get(handles.ShowDragMenu,'Checked')
    case 'on'
        tog_drag = 1;
    case 'off'
        tog_drag = 0;
end

%
%
%plot
if (get(handles.figure_check,'Value')) == 1
    figure(1)
    title('Projectile Motion Results')

else
    axes(handles.axes1);

end

hold on;

xlabel(['Distance ( ',unit_position,' )'])
ylabel(['Height ( ',unit_position,' )'])

%draw landing info and plot path
switch get(handles.ShowSimpleMenu,'Checked')
    case 'on'
        plot(x,y,'b*-')

        if tog_results == 1

            [a b] = size(x);

            text(x(b) , 0,[' ',num2str(x(b),'%9.3f'),...
                ' ',unit_position,' at ',num2str(hangtime,'%9.3f'),' seconds '],...
                'BackgroundColor',[.1 .7 .9],...
                'EdgeColor','k',...
                'HorizontalAlignment','center',...
                'VerticalAlignment','bottom')

```

```

        end

    case 'off'
        %nadda
    end

%plot drag
switch get(handles.ShowDragMenu,'Checked')
case 'on'
    if (get(handles.drag_complex,'Value')) == 1 | (get(handles.drag_simple,'Value')) == 1

        plot(x_drag, y_drag, 'g*-.')

        %draw results
        if tog_results == 1

            [a b] = size(x_drag);

            text(x_drag(b), -x_drag(b)/30, [' ', num2str(x_drag(b), '%9.3f'), ' ', unit_position, ' ', ...
                ' at ', num2str(hangtime_drag, '%9.3f'), ' seconds '], ...
                'BackgroundColor',[.1 .9 .5], ...
                'EdgeColor','k', ...
                'HorizontalAlignment','center', ...
                'VerticalAlignment','top')
        end
    end

end

case 'off'
    %nadda
end

%set axis to display results
axis auto

%draw results

axis equal;
user_axis = axis;

x_length = abs(user_axis(1) - user_axis(2)).*0.15;
y_length = abs(user_axis(3) - user_axis(4)).*0.15;

%set ground

```

```

user_axis(3) = 0;

%adjust axis
user_axis(4) = user_axis(4) * 1.1;
axis([user_axis])
%

%if draw results and this path
if tog_results == 1 & tog_simple == 1

    text(x_max - x_length*2.0, y_max + y_length*1.5, 'No Drag Ymax Data', 'HorizontalAlignment', 'center')
    text(x_max - x_length*2.0, y_max + y_length*1.0, [' X position = ', num2str(x_max, '%9.3f'), '
', unit_position, ' '], ...
        'BackgroundColor', [.8 .8 .8], ...
        'EdgeColor', 'k', ...
        'HorizontalAlignment', 'center', ...
        'VerticalAlignment', 'top')
    text(x_max - x_length*2.0, y_max + y_length*0.5, [' Y position = ', num2str(y_max, '%9.3f'), '
', unit_position, ' '], ...
        'BackgroundColor', [.8 .8 .8], ...
        'EdgeColor', 'k', ...
        'HorizontalAlignment', 'center', ...
        'VerticalAlignment', 'top')
    text(x_max - x_length*2, y_max + y_length*0, [' Time = ', num2str(time_max, 4), ' seconds '], ...
        'BackgroundColor', [.8 .8 .8], ...
        'EdgeColor', 'k', ...
        'HorizontalAlignment', 'center', ...
        'VerticalAlignment', 'top')

    plot(x_max, y_max, 'ro', ...
        'MarkerEdgeColor', 'k', ...
        'MarkerFaceColor', 'r', ...
        'MarkerSize', 10)
end

%check for drag values
if (get(handles.drag_complex, 'Value')) == 1 | (get(handles.drag_simple, 'Value')) == 1
    %already set value
    tog_drag = tog_drag;
else
    tog_drag = 0;
end

%if draw results and this path
if tog_results == 1 & tog_drag == 1
    text(x_drag_max - x_length*1.5, y_drag_max - y_length*1, 'Drag Path Ymax
Data', 'HorizontalAlignment', 'center')
    text(x_drag_max - x_length*1.5, y_drag_max - y_length*1.6, [' X position =
', num2str(x_drag_max, '%9.3f'), ' ', unit_position, ' '], ...

```

```

        'BackgroundColor',[.8 .8 .8],...
        'EdgeColor','k',...
        'HorizontalAlignment','center',...
        'VerticalAlignment','bottom')
    text(x_drag_max - x_length*1.5, y_drag_max - y_length*2.1,[' Y position =
',num2str(y_drag_max,'%9.3f'),' ',unit_position,' '],...
        'BackgroundColor',[.8 .8 .8],...
        'EdgeColor','k',...
        'HorizontalAlignment','center',...
        'VerticalAlignment','bottom')
    text(x_drag_max - x_length*1.5, y_drag_max - y_length*2.6,[' Time = ',num2str(time_drag_max,4), '
seconds '],...
        'BackgroundColor',[.8 .8 .8],...
        'EdgeColor','k',...
        'HorizontalAlignment','center',...
        'VerticalAlignment','bottom')

    plot(x_drag_max, y_drag_max, 'ro',...
        'MarkerEdgeColor','k',...
        'MarkerFaceColor','r',...
        'MarkerSize',10)
end

```

```

%if output data to command window
if (get(handles.data_check,'Value')) == 1

```

```

    %set units
    %determine units
    switch get(handles.UnitsEnglish,'Checked')
    case 'on'
        unit_velocity = 'feet / second';
    case 'off'
        unit_velocity = 'meters / second';
    end

```

```

figure(2)
clf reset;
hold on;

```

```

subplot(2,1,1)

```

```

hold on;
title('Velocity vs. X position')
xlabel(['Distance ( ',unit_position,' ) '])
ylabel(['Velocity ( ',unit_velocity,' ) '])

```

```

%calc vmag
%vx(1) = [];
%vy(1) = [];

vmag = sqrt(vx.^2 + vy.^2);

%x(1) = [];

plot(x, vmag, 'b')

%if drag exists, plot
if length(vx_drag) > 3

%    %format data to account for the swing
vx_drag(2) = vx_drag(1);
vy_drag(2) = vy_drag(1);

vx_drag(1) = [];
%    vx_drag(1) = [];
%
vy_drag(1) = [];
%    vy_drag(1) = [];
%    %vmag_drag(1) = [];
%
%    %set initial velocity data for plot
%    vx_drag(1) = vx(1);
%    vy_drag(1) = vy(1);
%    %
%    x_drag(1) = [];
%    x_drag(1) = [];
%    %x_drag(1) = [];
%
%    y_drag(1) = [];
%    y_drag(1) = [];
%    %y_drag(1) = [];

vmag_drag = sqrt(vx_drag.^2 + vy_drag.^2);

plot(x_drag, vmag_drag, 'g')

legend('No Drag','With Drag',4)
end

%hold off;

```

```

%calc angle of velocity
v_angle = atan2(vy, vx).* 180 ./ pi;
%v_angle(1) = [];

subplot(2,1,2)
hold on;

title('Theta vs. X position')
xlabel(['Distance ( ',unit_position,' ) '])
ylabel(['Theta ( degrees ) '])

plot(x, v_angle, 'b')


%if drag exists, plot
if length(vx_drag) > 3

%           %add velocity data for (2)
%   vx_drag(2) = ( vx_drag(1) + vx_drag(3))/2;
%   vy_drag(2) = ( vy_drag(1) + vy_drag(3))/2;

%calc angle of velocity
v_angle_drag = atan2(vy_drag, vx_drag).* 180 ./ pi;

plot(x_drag, v_angle_drag, 'g')

legend('No Drag','With Drag',1)

end

%output data as a table
clc

time = 0;

fid = fopen('EFD_BALLDROP_output.txt','wb');

fprintf(fid, '\n-----');
fprintf(fid, '\n----- Simple Model Data -----');
fprintf(fid, '\n\n   time       X           Y           Vx           Vy ');
fprintf(fid, '\n ');

%display units
fprintf(fid, '\n [ seconds ]      [ %s ]      [ %s ]', unit_position, unit_velocity);
fprintf(fid, '\n-----');
for i = 1:length(x)
    fprintf(fid, '\n %7.3f \t%9.3f \t%9.3f \t%9.3f \t%9.3f', time, x(i), y(i), vx(i), vy(i));

    time = time + dt;
end

```

```

fclose('all');

%print text file to HTML
web(['file:' which('EFD_BALLDROP_output.txt')],'-browser')

%if drag data
if length(vx_drag) > 3

    time = 0;

    fid = fopen('EFD_BALLDROP_output_DRAG.txt','wb');

    fprintf(fid,'n-----');
    fprintf(fid,'n----- Drag Model Data -----');
    fprintf(fid,'n\n   time          X          Y          Vx          Vy ');
    fprintf(fid,'n ');

    %display units
    fprintf(fid,'n [ seconds ]      [ %s ]      [ %s ]',unit_position,unit_velocity);
    fprintf(fid,'n-----');

    for i = 1:length(x_drag)
        fprintf(fid,'n %7.3f \t%9.3f \t%9.3f \t%9.3f \t%9.3f', time,
x_drag(i),y_drag(i),vx_drag(i),vy_drag(i));

        time = time + dt;
    end

    fclose('all');

    %print text file to HTML
    web(['file:' which('EFD_BALLDROP_output_DRAG.txt')],'-browser')
end

end

hold off;

% -----
function FileMenu_Callback(hObject, eventdata, handles)
% hObject    handle to FileMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% -----
function OpenMenuItem_Callback(hObject, eventdata, handles)
% hObject    handle to OpenMenuItem (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB

```



```

% handles structure with handles and user data (see GUIDATA)
file = uigetfile('*.fig');
if ~isequal(file, 0)
    open(file);
end

% -----
function PrintMenuItem_Callback(hObject, eventdata, handles)
% hObject handle to PrintMenuItem (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
printdlg(handles.figure1)

% -----
function CloseMenuItem_Callback(hObject, eventdata, handles)
% hObject handle to CloseMenuItem (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
selection = questdlg(['Close ' get(handles.figure1,'Name') '?'],...
    ['Close ' get(handles.figure1,'Name') '...'],...
    'Yes','No','Yes');
if strcmp(selection,'No')
    return;
end

delete(handles.figure1)

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton2 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton3 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton4 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton5 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

```

beep

```
% -----
function ViewMenu_Callback(hObject, eventdata, handles)
% hObject    handle to ViewMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% -----
function DisplayMenu_Callback(hObject, eventdata, handles)
% hObject    handle to DisplayMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% -----
function ErrorCheckMenu_Callback(hObject, eventdata, handles)
% hObject    handle to ErrorCheckMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% -----
function HelpMenu_Callback(hObject, eventdata, handles)
% hObject    handle to HelpMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% -----
function HelpGuideMenu_Callback(hObject, eventdata, handles)
% hObject    handle to HelpGuideMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

web(['file:' which('help_home.html')],'-browser')

% -----
function HelpAboutMenu_Callback(hObject, eventdata, handles)
% hObject    handle to HelpAboutMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

msgbox('Projectile Motion Analysis for Swing Launching,
this program','help')

% --- Executes during object creation, after setting all properties.
function listBox3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listBox3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

written by Jon Huber', 'About

```

% Hint: listbox controls usually have a white background on Windows.
%     See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

set(hObject, 'String', {'Joint', 'Member', 'Constraint', 'Force'});

% --- Executes on selection change in listbox3.
function listbox3_Callback(hObject, eventdata, handles)
% hObject    handle to listbox3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns listbox3 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from listbox3

% -----
function ZoomRotateMenu_Callback(hObject, eventdata, handles)
% hObject    handle to ZoomRotateMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in se_check.
function se_check_Callback(hObject, eventdata, handles)
% hObject    handle to se_check (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of se_check

% --- Executes during object creation, after setting all properties.
function se_x_CreateFcn(hObject, eventdata, handles)
% hObject    handle to se_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFns called

% Hint: edit controls usually have a white background on Windows.
%     See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function se_x_Callback(hObject, eventdata, handles)

```

```

% hObject   handle to se_x (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of se_x as text
%        str2double(get(hObject,'String')) returns contents of se_x as a double

% --- Executes during object creation, after setting all properties.
function se_y_CreateFcn(hObject, eventdata, handles)
% hObject   handle to se_y (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function se_y_Callback(hObject, eventdata, handles)
% hObject   handle to se_y (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of se_y as text
%        str2double(get(hObject,'String')) returns contents of se_y as a double

% --- Executes during object creation, after setting all properties.
function se_r_CreateFcn(hObject, eventdata, handles)
% hObject   handle to se_r (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function se_r_Callback(hObject, eventdata, handles)
% hObject   handle to se_r (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

```

```

% Hints: get(hObject,'String') returns contents of se_r as text
%      str2double(get(hObject,'String')) returns contents of se_r as a double

% --- Executes during object creation, after setting all properties.
function se_start_angle_CreateFcn(hObject, eventdata, handles)
% hObject    handle to se_start_angle (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%      See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function se_start_angle_Callback(hObject, eventdata, handles)
% hObject    handle to se_start_angle (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of se_start_angle as text
%      str2double(get(hObject,'String')) returns contents of se_start_angle as a double

% --- Executes during object creation, after setting all properties.
function se_release_angle_CreateFcn(hObject, eventdata, handles)
% hObject    handle to se_release_angle (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%      See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function se_release_angle_Callback(hObject, eventdata, handles)
% hObject    handle to se_release_angle (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of se_release_angle as text
%      str2double(get(hObject,'String')) returns contents of se_release_angle as a double

```

```

% --- Executes on button press in td_check.
function td_check_Callback(hObject, eventdata, handles)
% hObject    handle to td_check (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of td_check


% --- Executes during object creation, after setting all properties.
function prop_vt_CreateFcn(hObject, eventdata, handles)
% hObject    handle to prop_vt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


function prop_vt_Callback(hObject, eventdata, handles)
% hObject    handle to prop_vt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of prop_vt as text
%       str2double(get(hObject,'String')) returns contents of prop_vt as a double


% --- Executes during object creation, after setting all properties.
function prop_mass_CreateFcn(hObject, eventdata, handles)
% hObject    handle to prop_mass (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


function prop_mass_Callback(hObject, eventdata, handles)

```

```

% hObject    handle to prop_mass (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'String') returns contents of prop_mass as text
%    str2double(get(hObject,'String')) returns contents of prop_mass as a double


% --- Executes on button press in se_same.
function se_same_Callback(hObject, eventdata, handles)
% hObject    handle to se_same (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of se_same
set(handles.se_same, 'Value', 1);
set(handles.se_offset, 'Value', 0);
set(handles.se_specify, 'Value', 0);

% --- Executes on button press in se_offset.
function se_offset_Callback(hObject, eventdata, handles)
% hObject    handle to se_offset (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of se_offset
set(handles.se_same, 'Value', 0);
set(handles.se_offset, 'Value', 1);
set(handles.se_specify, 'Value', 0);

% --- Executes on button press in se_specify.
function se_specify_Callback(hObject, eventdata, handles)
% hObject    handle to se_specify (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of se_specify
set(handles.se_same, 'Value', 0);
set(handles.se_offset, 'Value', 0);
set(handles.se_specify, 'Value', 1);

% --- Executes during object creation, after setting all properties.
function prop_dt_CreateFcn(hObject, eventdata, handles)
% hObject    handle to prop_dt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%    See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));

```

end

```
function prop_dt_Callback(hObject, eventdata, handles)
% hObject   handle to prop_dt (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of prop_dt as text
%        str2double(get(hObject,'String')) returns contents of prop_dt as a double
```

```
% --- Executes during object creation, after setting all properties.
function se_offset_angle_CreateFcn(hObject, eventdata, handles)
% hObject   handle to se_offset_angle (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   empty - handles not created until after all CreateFns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function se_offset_angle_Callback(hObject, eventdata, handles)
% hObject   handle to se_offset_angle (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of se_offset_angle as text
%        str2double(get(hObject,'String')) returns contents of se_offset_angle as a double
```

```
% --- Executes during object creation, after setting all properties.
function se_specify_angle_CreateFcn(hObject, eventdata, handles)
% hObject   handle to se_specify_angle (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   empty - handles not created until after all CreateFns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```



```

function se_specify_angle_Callback(hObject, eventdata, handles)
% hObject   handle to se_specify_angle (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of se_specify_angle as text
%        str2double(get(hObject,'String')) returns contents of se_specify_angle as a double

% --- Executes during object creation, after setting all properties.
function td_xmin_CreateFcn(hObject, eventdata, handles)
% hObject   handle to td_xmin (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function td_xmin_Callback(hObject, eventdata, handles)
% hObject   handle to td_xmin (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of td_xmin as text
%        str2double(get(hObject,'String')) returns contents of td_xmin as a double

% --- Executes during object creation, after setting all properties.
function td_xmax_CreateFcn(hObject, eventdata, handles)
% hObject   handle to td_xmax (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function td_xmax_Callback(hObject, eventdata, handles)
% hObject   handle to td_xmax (see GCBO)

```

```

% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of td_xmax as text
% str2double(get(hObject,'String')) returns contents of td_xmax as a double

% --- Executes during object creation, after setting all properties.
function td_ymin_CreateFcn(hObject, eventdata, handles)
% hObject handle to td_ymin (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function td_ymin_Callback(hObject, eventdata, handles)
% hObject handle to td_ymin (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of td_ymin as text
% str2double(get(hObject,'String')) returns contents of td_ymin as a double

% --- Executes during object creation, after setting all properties.
function td_ymax_CreateFcn(hObject, eventdata, handles)
% hObject handle to td_ymax (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function td_ymax_Callback(hObject, eventdata, handles)
% hObject handle to td_ymax (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

```

```

% Hints: get(hObject,'String') returns contents of td_ymax as text
%      str2double(get(hObject,'String')) returns contents of td_ymax as a double


% --- Executes during object creation, after setting all properties.
function particle_mass_CreateFcn(hObject, eventdata, handles)
% hObject    handle to particle_mass (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFns called


% Hint: edit controls usually have a white background on Windows.
%      See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


function particle_mass_Callback(hObject, eventdata, handles)
% hObject    handle to particle_mass (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% Hints: get(hObject,'String') returns contents of particle_mass as text
%      str2double(get(hObject,'String')) returns contents of particle_mass as a double


% --- Executes during object creation, after setting all properties.
function drag_density_CreateFcn(hObject, eventdata, handles)
% hObject    handle to drag_density (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFns called


% Hint: edit controls usually have a white background on Windows.
%      See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


function drag_density_Callback(hObject, eventdata, handles)
% hObject    handle to drag_density (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% Hints: get(hObject,'String') returns contents of drag_density as text
%      str2double(get(hObject,'String')) returns contents of drag_density as a double

```

```

% --- Executes during object creation, after setting all properties.
function drag_k_CreateFcn(hObject, eventdata, handles)
% hObject    handle to drag_k (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function drag_k_Callback(hObject, eventdata, handles)
% hObject    handle to drag_k (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of drag_k as text
%        str2double(get(hObject,'String')) returns contents of drag_k as a double

% --- Executes on button press in drag_simple.
function drag_simple_Callback(hObject, eventdata, handles)
% hObject    handle to drag_simple (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of drag_simple
set(handles.drag_no, 'Value', 0);
set(handles.drag_simple, 'Value', 1);
set(handles.drag_complex, 'Value', 0);

% --- Executes on button press in drag_complex.
function drag_complex_Callback(hObject, eventdata, handles)
% hObject    handle to drag_complex (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of drag_complex
set(handles.drag_no, 'Value', 0);
set(handles.drag_simple, 'Value', 0);
set(handles.drag_complex, 'Value', 1);

% --- Executes on button press in drag_no.
function drag_no_Callback(hObject, eventdata, handles)
% hObject    handle to drag_no (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

% Hint: get(hObject,'Value') returns toggle state of drag_no
set(handles.drag_no, 'Value', 1);
set(handles.drag_simple, 'Value', 0);
set(handles.drag_complex, 'Value', 0);

% --- Executes during object creation, after setting all properties.
function drag_loss_CreateFcn(hObject, eventdata, handles)
% hObject    handle to drag_loss (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function drag_loss_Callback(hObject, eventdata, handles)
% hObject    handle to drag_loss (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of drag_loss as text
%       str2double(get(hObject,'String')) returns contents of drag_loss as a double

% --- Executes during object creation, after setting all properties.
function se_loss_CreateFcn(hObject, eventdata, handles)
% hObject    handle to se_loss (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function se_loss_Callback(hObject, eventdata, handles)
% hObject    handle to se_loss (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of se_loss as text

```

```
%    str2double(get(hObject,'String')) returns contents of se_loss as a double
```

```
% --- Executes on button press in figure_check.  
function figure_check_Callback(hObject, eventdata, handles)  
% hObject    handle to figure_check (see GCBO)  
% eventdata  reserved - to be defined in a future version of MATLAB  
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hint: get(hObject,'Value') returns toggle state of figure_check
```

```
% -----  
function ShowSimpleMenu_Callback(hObject, eventdata, handles)  
% hObject    handle to ShowSimpleMenu (see GCBO)  
% eventdata  reserved - to be defined in a future version of MATLAB  
% handles    structure with handles and user data (see GUIDATA)
```

```
switch get(handles.ShowSimpleMenu,'Checked')  
    case 'on'  
        set(handles.ShowSimpleMenu, 'Checked', 'off')  
    case 'off'  
        set(handles.ShowSimpleMenu, 'Checked', 'on')  
end
```

```
% -----  
function ShowDragMenu_Callback(hObject, eventdata, handles)  
% hObject    handle to ShowDragMenu (see GCBO)  
% eventdata  reserved - to be defined in a future version of MATLAB  
% handles    structure with handles and user data (see GUIDATA)
```

```
switch get(handles.ShowDragMenu,'Checked')  
    case 'on'  
        set(handles.ShowDragMenu, 'Checked', 'off')  
    case 'off'  
        set(handles.ShowDragMenu, 'Checked', 'on')  
end
```

```
% -----  
function UnitsMenu_Callback(hObject, eventdata, handles)  
% hObject    handle to UnitsMenu (see GCBO)  
% eventdata  reserved - to be defined in a future version of MATLAB  
% handles    structure with handles and user data (see GUIDATA)
```

```
% -----  
function UnitsEnglish_Callback(hObject, eventdata, handles)  
% hObject    handle to UnitsEnglish (see GCBO)  
% eventdata  reserved - to be defined in a future version of MATLAB  
% handles    structure with handles and user data (see GUIDATA)
```

```

set(handles.UnitsEnglish, 'Checked', 'on');
set(handles.UnitsMetric, 'Checked', 'off');

%pop-up figure with units menu
figure(2);
clf reset;
axis([0 100 0 100])

title('English System of Units')

text(5,96,'Properties')
text(5,93,'-----')
text(5,90,'dt == ( sec )')
text(5,85,'Vt == ( ft / sec )')
text(5,80,'mass == ( slugs )')

text(5,71,'Calculate Swing Energy')
text(5,68,'-----')
text(5,65,'X == ( ft )')
text(5,60,'Y == ( ft )')
text(5,55,'radius == ( ft )')
text(5,50,'start angle == ( degrees )')
text(5,45,'release angle == ( degrees )')
text(5,40,'loss == ( value of 0 thru 100 )')

text(5,31,'Particle Release Angle')
text(5,28,'-----')
text(5,25,'Offset == ( degrees )')
text(5,20,'Specify == ( degrees )')
text(5,15,'mass of projectile == ( slugs )')

text(55,96,'Aim for Target')
text(55,93,'-----')
text(55,90,'Xmin, Xmax == ( m )')
text(55,85,'Ymin, Ymax == ( m )')

text(55,61,'Drag Menu')
text(55,58,'-----')
text(55,55,'loss == ( value of 0 thru 100 )')
text(55,50,'k == ( lb * s / ft )')
text(55,45,'density of air == ( slug / ft^3 )')

% -----
function UnitsMetric_Callback(hObject, eventdata, handles)
% hObject    handle to UnitsMetric (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

set(handles.UnitsEnglish, 'Checked', 'off');
set(handles.UnitsMetric, 'Checked', 'on');

```

```

%pop-up figure with units menu
figure(2);
clf reset;
axis([0 100 0 100])

title('Metric System of Units')

text(5,96,'Properties')
text(5,93,'-----')
text(5,90,'dt == ( sec )')
text(5,85,'Vt == ( m / sec )')
text(5,80,'mass == ( kg )')

text(5,71,'Calculate Swing Energy')
text(5,68,'-----')
text(5,65,'X == ( m )')
text(5,60,'Y == ( m )')
text(5,55,'radius == ( m )')
text(5,50,'start angle == ( degrees )')
text(5,45,'release angle == ( degrees )')
text(5,40,'loss == ( value of 0 thru 100 )')

text(5,31,'Particle Release Angle')
text(5,28,'-----')
text(5,25,'Offset == ( degrees )')
text(5,20,'Specify == ( degrees )')
text(5,15,'mass of projectile == ( kg )')

text(55,96,'Aim for Target')
text(55,93,'-----')
text(55,90,'Xmin, Xmax == ( m )')
text(55,85,'Ymin, Ymax == ( m )')

text(55,61,'Drag Menu')
text(55,58,'-----')
text(55,55,'loss == ( value of 0 thru 100 )')
text(55,50,'k == ( N * s / m )')
text(55,45,'density of air == ( kg / m^3 )')

% -----
function findme_Callback(hObject, eventdata, handles)
% hObject    handle to findme (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% -----
function ShowResultsMenu_Callback(hObject, eventdata, handles)
% hObject    handle to ShowResultsMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```



```

switch get(handles.ShowResultsMenu,'Checked')
    case 'on'
        set(handles.ShowResultsMenu, 'Checked', 'off')
    case 'off'
        set(handles.ShowResultsMenu, 'Checked', 'on')
end

% --- Executes on button press in data_check.
function data_check_Callback(hObject, eventdata, handles)
% hObject    handle to data_check (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of data_check

```

Vita

I started my thesis journey in 1998. I spent two years in graduate school at the University of Tennessee, and then left to take a job as a design engineer for a third tier automotive manufacturer. They produce airbag inflators (glorified pipebombs).

Unfortunately, I never completed my thesis. However, I was washing away the green of my newness. During this time, I realized I needed my graduate degree. I went part time for my job and went back to being the lowly graduate assistant for the *engage* freshmen engineering program. I enjoy my additional year as a TA because I am considering a full academic professional lifestyle.

For the future, I plan on returning to the automotive company where I will perform stress analysis on the inflators as well as write analysis code to model the explosive material and airflow.

I am looking forward to a life within science. I have every intention to continue living with a thirst for knowledge and understanding.