Computer Aided Instruction Package

on

Computer Problems for Engineering Mechanics

Sponsored by

AICTE
under Curriculum Development Program

Author
Dr. G. Saravana Kumar

Department of Mechanical Engineering
Indian Institute of Technology Guwahati
Guwahati - 781 039.

October 2008
The author has taken care in preparation of this documentation but makes no expressed or implied warranty of any kind and assumes no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising from the use of information or programs contained herein.

Copyright © 2008, G. Saravana Kumar. All rights reserved.

All rights reserved. No part of this documentation may be reproduced, in any form or by any means, without permission in writing from the author.
PREFACE

The Quality Improvement Programme (QIP) supported by the Ministry of Human Resource Development (MHRD) Government of India through All India Council for Technical Education (AICTE) aims at developing better teaching capabilities within the country. The main vehicle of this program is the development of curriculum and training of the teachers for the same. This manual and accompanying CD for Computer Aided Instruction Package on “Computer Problems for Engineering Mechanics” has been developed under the Curriculum Development Program, Indian Institute of Technology Guwahati (IITG), Assam.

This computer aided instruction package for Engineering Statics has been developed with the aid of high-level programming tool Matlab®. The programs are designed not to be standalone teaching resources, replacing functions that lecturers and textbooks do well, but to address the particular problem of providing interactive help in an environment of high student/staff ratios. Their main function is to interactively solve problems with the student, but additionally, various means are used to reinforce the theory presented in lectures. The aim is to integrate tightly with the other elements of the course, and not add to the burden on the student by introducing new material.

This manual contains the materials for understanding, running the computer aided instruction package for Engineering Statics that is available in the accompanying CD. Specific topics on which the programs have been provided are equilibrium of particles, equilibrium of rigid bodies, equilibrium of Trusses, equilibrium and shear force and bending moments in Beams, equilibrium in the presence of friction, method of virtual work, computing area moments and mass moment of inertia. This computer aided instruction package has been prepared by the author who has taught this course at the under graduate level in the capacity of the instructor as well as tutor for four years in the Department of Mechanical Engineering, IITG.

Dr. G. Saravana Kumar
Assistant Professor
Department of Mechanical Engineering
Indian Institute of Technology Guwahati
Guwahati – 781039.
ACKNOWLEDGEMENT

The author of this Computer Aided Instruction Package on “Computer Problems for Engineering Mechanics” under Curriculum Development Program is thankful to the Ministry of Human Resource Development (MHRD) Government of India for providing financial support to prepare the software and manual through AICTE. The author also expresses his gratitude to Prof. Rajiv Tiwari, Coordinator, QIP, IITG for providing his perennial support throughout the preparation of this courseware. He is also thankful to his colleagues, Prof. Uday S Dixit, Department of Mechanical Engineering, IITG, Dr. Arvind Kumar Singh, Department of Civil Engineering, IITG who provided many helpful comments and suggestions. He also thanks Prof. B Gurunathan, Retired Professor, Department of Mechanical Engineering, A C College of Engineering and Technology, Karaikudi who meticulously corrected many errors in this document and also provided useful comments and suggestions for the programs.

Dr. G. Saravana Kumar
Assistant Professor
Department of Mechanical Engineering
Indian Institute of Technology Guwahati
Guwahati – 781039.
CONTENTS

Preface
Acknowledgement

1. Introduction
   1.1 Background
   1.2 Design Philosophy
   1.3 Conventions and Terminology

2. GUIs for Computer Problems in Statics
   2.1 Equilibrium of Particles in 2D
   2.2 Equilibrium of Rigid Body in 2D
   2.3 Equilibrium of Trusses in 2D (Forces in members)
   2.4 Equilibrium of Beams in 2D (Shear Force and Bending Moment Diagrams)
   2.5 Equilibrium considering Friction
   2.6 Principle of Virtual Work
   2.7 Properties of Area
   2.8 Properties of Volume

3. Usage Guidelines

4. Implementation Issues
   o 4.1 Name Space Conventions
   o 4.2 Modular Implementation
   o 4.3 Error Checking and Reporting

A Matlab GUI files
B Matlab Library Function files

References
1. INTRODUCTION

The ‘Computer Problems for Engineering Mechanics’ toolkit consists of Graphical User Interfaces (GUIs) and supporting libraries which are a set of programming interface with Matlab® programming environment. The toolkit supports in the current version the following functionality:

- Call back driven GUI’s that allow mechanics problems to be defined, input suitable problem statement, figure files and input data as a data structure file.
- Utility libraries for solving many statics problems including
  - Equilibrium of Particles (2D)
  - Equilibrium of Rigid Bodies (2D)
  - Equilibrium of Trusses (Forces in Members)
  - Equilibrium of Beams (Shear Force and Bending Moment Diagrams)
  - Equilibrium in the presence of Friction
  - Equilibrium analysis using method of Virtual Work
  - Properties of Areas: C.G, moments
  - Properties of Volumes: C.G, moments

- Example problems

An Matlab® implementation for the Windows Version 7.3.0.267 (R2006b) has been implemented by the author. Although not fully tested on older versions, they seem to be backwards compatible. The accompanying CD contains all the above files in structured directories.

This documentation serves as both a programming guide in using the library of programs developed and hard copy repository of the libraries. Teachers and students interested in using the toolkit in conjunction with a college-level mechanics course should first have a basic understanding of the Matlab® programming environment. If the reader is interested in introduction to Matlab® programming, he is urged to look for the introductory texts [1,2] available.

The remainder of this section describes the tool kit design philosophy and usage model. The following sections specify the GUI routines, grouped chapter wise as occurring in most standard texts in Statics [3-6]. One example problem has also been solved for each case. The final section discusses usage advice, implementation issues. Appendix A presents the toolkit GUI’s and related problem input files via the Matlab® program files. Appendix B presents the toolkit libraries via Matlab® program files.

1.1 Background

This computer aided instruction package for Engineering Statics has been developed with the aid of high-level programming tool Matlab®. The programs are designed not to be standalone teaching resources, replacing functions that lecturers and textbooks do well, but to address the particular problem of providing interactive help in an environment of high student/staff ratios. Their main function is to interactively solve problems with the student, but additionally, various means are used to reinforce the theory presented in lectures. The aim is to integrate tightly with
the other elements of the course, and not add to the burden on the student by introducing new material.

Many textbooks on *Engineering Statics* provide computer problems. The books provide for each chapter a set of problems that have been designed to be solved with computational software. Many of these problems provide an introduction to the design process. In *Statics*, for example, they may involve analysis of a structure for various configurations of loading of the structure or the determination of the equilibrium positions of mechanisms, which may require an iterative method of solution. Developing an algorithm to solve a given mechanics problem will benefit the student in two ways: (1) it will help them gain a better understanding of the mechanics principle involved; (2) it will provide them with an opportunity to apply their computer skills to the solution of a meaningful engineering problem.

This toolkit is designed to fill this need. The toolkit provides a Matlab® dependent programming interface for mechanics problems. The teacher can use this to demonstrate few problems to the students in the class and the student may be encouraged to develop many more similar problems and contribute to the library of problems and also expand the toolkit library of user defined functions in this process.

![Fig.1. General layout of the GUI’s](image)

1.2 Design Philosophy
The GUIs developed in the toolkit simplify the implementation of programs for solving computer problems in *Statics*. Most initial state of the window is defined and this is reasonable for simple programs. The general layout of the GUIs created is shown in Fig. 1. It consists of a predominant graphical window, a problem statement window below and the right hand side constituting of toolbox containing menu buttons. The graphical window is used too render the figure files (*.tiff), problem specific plots and diagrams. The problem statement window is used to display the problem statement (*.txt) files, the solution steps and final solution as the appropriate buttons in toolbox are pressed. The toolbox contains a pull down menu to choose a class of problem, a pull down menu to modify the data, push buttons to solve, provide info and close the GUI window. The ‘pull down’ menus have callback routines to import and modify the data as well as supply figure and text data to the graphical window and problem statement window respectively. The ‘solve’ button’s callback routine to solve the problem and supply text as well figure to the graphical window and problem statement window respectively. The ‘info’ button’s callback routine to display help in Matlab® help window. The ‘close’ button’s callback routine kills the window and all the associated data.

1.3 Conventions and Terminology

The GUI’s graphical window when used to render image data like that from '*.tiff’ need special attention. The image coordinates are expressed in pixels and the upper left hand corner in (0,0), X coordinate increases rightward direction. The Y coordinate increases in the downward direction. This is inconsistent with the graphical coordinate system and thus suitable transformations have been incorporated while displaying image files.

The results of computation are stored as floating-point numbers with double precision (max 16 decimal places). Thus all the results displayed are to be considered to three decimal places only for engineering significance.

A number of terms are used in a Matlab® GUI specific manner throughout this document. The meanings of some specific terms are:

**Callback** A programmer specified routine that can be registered with a GUI entity like buttons to be called in response to a specific type of window event. Also used to refer to a specific routine being called.

**Tool Box** A sophisticated input device consisting of pad of push buttons and array of pull down menu, radio buttons.

**Pop-up menu** A menu that can be set to appear when a specified mouse button is pressed in a GUI button. A pop-menu consists of multiple menu items.

**Reshape** The act of changing the size or shape of the window.

**Window** A rectangular area for GUI rendering.

2. **GUIS FOR COMPUTER PROBLEMS IN STATIC**
2.1 Equilibrium of Particles in 2D

**Description:** The equilibrium2DP is a GUI in Matlab® that Demonstrates 2D equilibrium of particles. This window allows the User to input a 2D equilibrium problem (a text file describing the problem, coordinates, units and a figure file showing the structure for the 2D equilibrium problem) and provides the required solution. By playing with the popup menus on the right side of the window, the User can adjust the type of problem, the solutions required. The MiniCommand Window in the lower right shows the problem descriptions and also solution values. Fig. 2 shows a snap shot of the GUI with example problem1.

**Usage:** equilibrium2DP in MATLAB command prompt. Create new problem by creating
- Suitable Problem*.txt file, Problem*As Given.tiff and Problem*Modify.tiff (corresponding to as given data and modification).
- Edit line 123 to add Problem* Label
- Add additional if else loop for displaying problem* image with origin (line 225)
- Add suitably for problem specific data in the if else loop at line 239
- Add suitably solution procedure in the switch at line 288.

**Example Problem:**
Problem text File:- Problem1.txt
The direction of the 75-N forces may vary, but the angle between the forces is always 50°. Determine the value of alpha for which the resultant of the forces acting at A is directed horizontally to the left.

Figure File:- Problem1As Given.tiff

**Solution:**
Forces are represented as [Xmag Ymag Xpos Ypos] in N
Forces=matrix([-sqrt(43200),120,0,0],[75*cos(1/180*t*pi),75*sin(1/180*t*pi),0,0],[75*cos(1/180*(t+50)*pi),75*sin(1/180*(t+50)*pi),0,0])
Equate the forces to zero and Solve for t
Forces=[150 240 0 0;-86.97 75 0 0;-36.97 75 0 0]
Resultant=[180 143.96 0 0]
Alpha=-86.97
2.2 Equilibrium of Rigid Body in 2D

**Description:** The equilibrium2DR is a GUI in Matlab® that Demonstrates 2D equilibrium of rigid body. This window allows the User to input a 2D equilibrium problem (a text file describing the problem, coordinates, units and a figure file showing the structure for the 2D equilibrium problem) and provides the required solution. By playing with the popup menus on the right side of the window, the User can adjust the type of problem, the solutions required. The MiniCommand Window in the lower right shows the problem descriptions and also solution values. Fig. 3 shows a snap shot of the GUI with example problem1.
Fig. 3 equilibrium2DR GUI in Matlab®

Usage: equilibriu2DR in MATLAB command prompt. Create new problem by creating
- Suitable Problem*.txt file, Problem*As Given.tiff and Problem*Modify.tiff (corresponding to as given data and modification).
- Edit line 120 to add Problem* Label
- Add additional if else loop for displaying problem* image with origin (line 222)
- Add suitably for problem specific data in the if else loop at line 236
- Add suitably solution procedure in the switch at line 276.

Example Problem:
Problem text File:- Problem1.txt
What is the moment about A of the 500 N force and the 3000 N –m couple acting on the cantilever beam?

Figure File: - Problem1As Given.tiff

Solution:-
Forces are represented as [Xmag Ymag Xpos Ypos] in N
Moments are represented as [mag Xpos Ypos (flag for Couple)] in Nm
Forces=[-433.01 -250 5 0]
Moments=[3000 0 0 1]
Solve for Resultant at A
2.3 Equilibrium of Trusses in 2D (Forces in members)

Description: Truss2D is a GUI in Matlab® to demonstrate 2D equilibrium of Truss (Connected Rigid Bodies). This window allows the user to input a 2D Truss problem (a data text file describing the problem, coordinates, units) and provides the required solution (Forces in Members, Support Reactions). By playing with the popup menus on the right side of the window, the user can adjust the geometry, type of reactions and applied forces on the truss, the solutions required. The MiniCommand Window in the lower right shows the problem descriptions and also solution values. Fig. 4 shows a snap shot of the GUI with example problem1.

Usage: Truss2D in MATLAB command prompt. Create new truss problem by creating
- Suitable Problem1.txt and data in Problem1.m file

Example Problem:
Problem text File: - Problem1.txt
A ten-bar truss is considered as shown in the figure. The truss is pin connected at the nodes 5 and 6. Determine the forces in all the members. Determine the Support Reactions.

Solution:
Coordinates of Nodes=[720 360 0; 720 0 0; 360 360 0; 360 0 0; 0 360 0; 0 0 0]

Members=[5 3; 1 3; 6 4; 4 2; 4 3; 2 1; 6 3; 5 4; 4 1; 3 2]

Degrees of Freedom (X, Y, Z and O - free; 1- fixed)[0 0 1; 0 0 1; 0 1; 0 1 1; 1 1 1]

Forces are represented as [Xmag Ymag Xpos Ypos] in N

Applied Forces=[0 0 720 360; 0 -100000 720 0; 0 0 360 360; 0 -100000 360 0; 0 0 0 360; 0 0 0 0]

Solve by first finding the reactions and then the forces in members by Method of Joints

Forces (N) in Members 1 2...n
Force=[131991.83 69625.84 -268008.16 -30374.15 1617.68 69625.84 -45243.28 237599.42 -98465.81 42955.54]

Reaction (N) at Supports (Matrix with n*3 (n = No. of Joints))
Reaction=[0 0 0; 0 0 0; 0 0 0; -300000 168008.16 0; 0 0 0; 0 300000.00 31991.83 0]

2.4 Equilibrium of Beams in 2D (Shear Force and Bending Moment Diagrams)

Description: Beams is a GUI in Matlab® that Demonstrate 2D equilibrium of User Defined Beams. This window allows the user to input a 2D Beam problem (a data text file describing the beam, end conditions, loading) and provides the required solution (Reactions, Shear Force Diagram, Bending Moment Diagram). By playing with the popup menus on the right side of the window, the User can adjust the geometry, type of reactions and applied forces and couple moments on the beam, the solutions required. The MiniCommand Window in the lower right shows the problem descriptions and also solution values. Fig. 5 shows a snap shot of the GUI with example problem

Usage: Beams in MATLAB command prompt. Create new beam problem by creating
  * Suitable Problem1.txt and data in Problem1.m file

Example Problem:
Problem text File: -Problem1.txt
A Beam 20 m long is supported by a pin at 5m and by a roller at 19m from right end. The beam is acted upon by a concentrated load of 10N at right end and a distributed load of 30 N for 3m after 10m from the right end. A couple of 30 Nm acts at the right end. Determine the reactions and draw shear force and bending moment diagrams.
Solution: -

Beam Span=20
Forces are represented as [Xmag Ymag Xpos Ypos] in N
Moments in Nm
Reaction Unknown=[1.57 5 0;1.57 19 0;0 5 0]
Applied Forces=[0 -10 0 0;0 -90 11.5 0]
Applied Moment if any=30
Moment Location if any=0
Distributed Force if any=[-30 -30 3 10]
Unknown reactions represented as direction and location
Distributed Force as [Mag1 Mag2 span location]
Determine Reactions and Use differential relations between load, shear and bending moment and integrate as per limits
Reactions=[0 59.64 5 0; 0 40.35 19 0; 0 0 5 0]
MaxShear(N)=49.64
MaxMoment (Nm)=272.23
MaxMoment Location (from right) in m =11.7

![Graphs showing shear and moment](image)

Fig.5. Beams GUI in Matlab®

2.5 Equilibrium considering Friction

**Description:** Friction is a GUI in Matlab® to Demonstrate the effect of Co-efficient of Friction on the equilibrium of Rigid Bodies. This window allows the User to input a problem (image as well as data) and obtain results (such as the range of values of equilibrium position for
a range of co-efficient of Friction). By playing with the popup menus on the right side of the window, the User can adjust the geometry, type of reactions and applied forces and couple moments on the beam, the solutions required. The MiniCommand Window in the lower right shows the problem descriptions and also solution values. Fig. 6 shows a snap shot of the GUI with example problem1.

Usage: Friction in MATLAB command prompt. Edit problem by creating
- Suitable Problem1.txt, Problem1As Given.tiff and Problem1Modify.tiff image files and data in Problem1.m file
- Can create additional types of problem by creating addition switch structures as described in section 2.1, 2.2.

Example Problem:
Problem text File: - Problem1.txt
Two Masses m1=45kg and m2=36kg are tied by a rope over the pulley as shown in the figure. If the co-efficient of friction between block and sliding surface varies between 0 and 0.6. Determine the values of theta for which the motion is impending. Plot these values as a function of co-efficient of friction.

Figure File: - Problem1As Given.tiff

Solution: -

Fig.6. Friction GUI in Matlab®
Masses $m_1$ and $m_2$ in kg = [45 36]
Range of Co-efficient of Friction = [0 0.6]
Solve by creating 2 FBD and setting up the corresponding equilibrium equations.

The Positive values for theta are for the intended motion to right

The Negative values for theta are for the intended motion to left (consider the mirror of the Block Placement)

2.6 Principle of Virtual Work

**Description:** Virtualwork is a GUI in Matlab® to Demonstrate the use of the principle of Virtual Work in analyzing equilibrium of system of Rigid Bodies. This window allows the User to input a Problem (image as well as data as a data file) and provides results like range of equilibrium position for combination of forces and position of a system. By playing with the popup menus on the right side of the window, the User can adjust the geometry, type of reactions and applied forces and couple moments on the beam, the solutions required. The MiniCommand Window in the lower right shows the problem descriptions and also solution values. Fig. 7 shows a snap shot of the GUI with example problem1.

**Usage:** Virtualwork in MATLAB command prompt. Create new equilibrium problem by creating
- Suitable Problem1.txt, Problem1As Given.tiff and Problem1Modify.tiff image files and data in Problem1.m file
- Can create additional types of problem by creating addition switch structures as described in section 2.1, 2.2.

**Example Problem:**
Problem text File: - Problem1.txt
Knowing that \( a = 500 \text{ mm} \), \( b = 150 \text{ mm} \), \( L = 500 \text{ mm} \), and \( P = 110\text{N} \), use a program to plot the force in member BD as a function of theta from 30 deg to 150 deg. Determine the range of values of theta for which the absolute value of the force in member BD is less than 450 N.

Figure File: - Problem1As Given.tiff

![Diagram showing forces and variables]

Solution: -

The Force \( P \) in \( \text{N} \) = 110
The lengths AC, BC and DC in mm = [500 150 500]
Range of Values of Theta in degrees = [30 150]
Limiting magnitude of Force in member DB in \( \text{N} \) = 450
The Force in member BD for complete range of values of Theta is plotted in YELLOW (dotted line)
The Range of values of Theta for a limiting value of Force in BD is plotted in MAGENTA (continuous line)
The Range of values of Theta for a limiting value of Force in BD is = [35.72 133.13]
2.7 Properties of Area

**Description:** AreaMoments is a GUI in Matlab® that computes the area, centroidal coordinates, and inertial moments of an arbitrary user defined polygon. This window allows the User to input 2D Arbitrary Area (a data text file describing the corner coordinates of the Area). By playing with the popup menus on the right side of the window, the User can adjust the geometry, type of reactions and applied forces and couple moments on the beam, the solutions required. The MiniCommand Window in the lower right shows the problem descriptions and also solution values. Fig. 8 shows a snap shot of the GUI with example problem1.

**Usage:** AreaMoments in MATLAB command prompt. Create new geometry by creating
- Suitable Problem1.txt and data in Problem1.m file

**Example Problem:**
Problem text File: - Problem1.txt
For the given geometry determine the area, centroid and second moments.

Solution:

Area=116
Centroid =[5.19 6.63]
Second Moments XX, XY and YY=[3999.5 3902.08 6707.83]
2.8 Properties of Volume

**Description:** MassMoments is a GUI in Matlab® to Demonstrate the Computation of Volume, Mass, Centroidal coordinates, and Inertia Tensor of an arbitrary 3D polyhedron. This window allows the user to input 3D Arbitrary Polyhedron (a data text file describing the corner coordinates, faces of the Polyhedron). By playing with the popup menus on the right side of the window, the User can adjust the geometry, type of reactions and applied forces and couple moments on the beam, the solutions required. The MiniCommand Window in the lower right shows the problem descriptions and also solution values. Fig. 9 shows a snap shot of the GUI with example problem1.

**Usage:** MassMoments in MATLAB command prompt. Create new 3D geometry by creating
- Suitable Problem1.txt and data in Problem1.m file

**Example Problem:**
Problem text File: - Problem1.txt
For the given 3D geometry determine the volume, mass, centroid and second moments (Inertia Tensor).

**Solution:** -

Volume=15, Centroid = [0;2.66;1.33]
Second Moments of Volume $[XX, XY XZ; YX YY YZ; ZX ZY ZZ]=[5 0 0; 120.83 60.41 60.41; 0 40.83 40.83]$

Inertia Tensor for Unit Mass $[XX, XY XZ; YX YY YZ; ZX ZY ZZ]=[161.66 0 0; 45.83 -60.41 0; -60.41 125.83 0]$

Fig. 9. MassMoments GUI in Matlab®

3. USAGE GUIDELINES

The accompanying CD contains all the files required to run the various GUI's in structured directories. The user may, and should modify the M-files as they see fit. These were developed using MATLAB 7.3.0.267 (R2006b). Although not fully tested on older versions, they seem to be backwards compatible.

DISCLAIMER:
"These M-Files are User Contributed Routines which are upon request distributed on an "as is" basis. A User Contributed Routine is not a product of The Math Works, Inc. and The Math Works or the author assumes no responsibility for any errors that may exist in these routines."

There are a number of points to keep in mind when using and editing these GUI programs. Some of there are strong recommendations, others simply hints and tips.

- Do not change the state that will affect the way the GUI window will be drawn back using the callback routines.
- Avoid using Matlab® figure toolbars in the GUI window.
- Do not delete any GUI entity, axes, GUI buttons etc.
• The program initially reshapes the GUI window to the screen size and latter the user can manually resize as per the requirement.
• Do not call a callback routine by pressing a GUI button while the other callback routine is still under process. This case occurs since there are no warnings generated.
• On slow machines, slow rendering, computations may compromise menu performance.
• Keep in mind that even if a single entity in the GUI window is damaged, all the callback routines may be damaged and one has to restart the GUI.
• While reprogramming if you encounter warnings or fatal error in your programs, try setting a debugger breakpoint to determine where, within the program, the error occurred.

4. IMPLEMENTATION ISSUES

4.1 Name Space Conventions

The GUI implementation should and has a well-defined name space convention. There are many distinct types of files included:

• The demo files: ####.m
  o These are the GUI files that the user runs to see the demonstration. Refer Appendix A.

• The function files: library\####.m
  o These comprise the user defined library files. The MATLAB path has to be set to library. Refer Appendix B.

• The problem files: ####\Problem#.m
  o These files are used to provide input data as user defined data structure. Refer Appendix A.

• The problem files: ####\Problem#.txt
  o These files are used to provide the problem statement.

• The problem files: ####\Problem####.tiff
  o These files are image files used to provide the problem figures.

Users should add and modify files according to the above name conventions.

4.2 Modular Implementation

It is often the case that the GUI programs and the required libraries tend to result in large, bulky programs because a large measure of “dynamically dead” code is linked in to the programs because it can not be determined at link time that the program will never require (that is, execute) the code. A consideration (not a primary one though) in the GUI design is to make the GUI interface modular enough that programs using limited subset of user-defined functions can
minimize the portion of the implementation required. Some user-defined functions have been adapted from [7].

### 4.3 Error Checking and Reporting

How errors and warnings about the GUI usage are reported is implementation dependent. The recommended behavior in the case of the error is to output a message and exit. In the current implementation, the errors are displayed in the Matlab® command window but the GUI does not exit itself. The user has to exit the window and as per the error message debug the code or correct the input files. All improper usage of GUI’s have not been caught and reported. If the user deletes any entity in the GUI, no error message occurs. The run-time overhead of error checking for a very common operation may outweigh the benefit of clean error reporting. The trade-off is left for the User to make.
function equilibrium2DP(action)
% equilibrium2DP is a GUI in Matlab that Demonstrates 2D equilibrium of particles.
% This window allows the User to input a 2D equilibrium problem (a text file
% describing the problem, coordinates, units and a figure file showing the
% structure for the 2D equilibrium problem and provides the required
% solution.
%
% By playing with the popup menus on the right side of the window, the User can
% adjust the type of problem, the solutions required and so on.
%
% The MiniCommand Window in the lower right shows the problem descriptions
% and also solution values.
%
% User m functions called: deg2xy.m, xy2deg.m, showvect.m, showx.m, showy.m,
% sumforce.m and twovector.m
%
% The user may, and should modify the M-files as they see fit.
%
% These were developed using MATLAB 7.3.0.267 (R2006b). Although not
% fully tested on older versions, they seem to be backwards
% compatible.
%
% DISCLAIMER:
% "These M-Files are User Contributed Routines which are upon request
% distributed on an "as is" basis. A User Contributed Routine is not a
% product of The Math Works, Inc. and The Math Works or the author assumes
% no responsibility for any errors that may exist in these routines."
%
% Author : G Saravana Kumar, Year Created: 2008
% All rights reserved with the Author.
% *-----------------------------------------------------------------------

if nargin<1,
    action='initialize';
end;
if strcmp(action,'initialize'),
    oldFigNumber=watchon;
    scrsz = get(0,'ScreenSize');
    figNumber=figure( ...
    'Visible','off', ...
    'NumberTitle','off', ...
    'Color','black', ...
    'Name','2-D Equilibrium of Pacitcles', 'Position',scrsz);
    colordef(figNumber,'none')
    axes( ...
    'Units','normalized', ...
    'Position',[0.07 0.45 0.60 0.50]);

% Set up the Problem statement Window

% First, the Problem statement frame
frmBorder=0.02;
frmPos=[left-frmBorder bottom-frmBorder ...
(right-left)+2*frmBorder bottom-frmBorder ...
(right-left)+2*frmBorder (top-bottom)+2*frmBorder];
uitable( ...
'Style','frame', ...
'Units','normalized', ...
'Position',frmPos, ...)
% Then the text label
labelPos=[left top-labelHt (right-left) labelHt];
uicontrol(...
'Style','text', ...
'Units','normalized', ...
'Position',labelPos, ...
'BackgroundColor',[0.50 0.50 0.50], ... 
'ForegroundColor',[1 1 1], ...
'String','Problem Statement');
% Then the editable text field
mcwPos=[left bottom (right-left) top-bottom-labelHt-spacing]
mcwHndl=uicontrol(...
'Style','edit', ...
'HorizontalAlignment','left', ...
'Units','normalized', ...
'Max',10,...
'BackgroundColor',[1 1 1], ... 
'Position',mcwPos);
% Save this handle for future use
set(gcf,'UserData',mcwHndl);
%====================================
% Information for all buttons
labelColor=[0.8 0.8 0.8];
top=0.95;
bottom=0.05;
left=0.75;
yInitLabelPos=0.90;
left=0.75;
labelWid=0.20;
labelHt=0.05;
btnWid=0.20;
btnHt=0.05;
% Spacing between the label and the button for the same command
btnOffset=0.003;
% Spacing between the button and the next command's label
spacing=0.05;
%====================================
% The CONSOLE frame
frmBorder=0.02;
yPos=0.05-frmBorder;
frmPos=[left frmBorder yPos btnWid+2*frmBorder 0.9+2*frmBorder];
h=uicontrol(...
'Style','frame', ...
'Units','normalized', ...
'Position',frmPos, ...
'BackgroundColor',[0.50 0.50 0.50]);
%====================================
% The PROBLEM INPUT command popup button
btnNumber=1;
yLabelPos=top-(btnNumber-1)*(btnHt+labelHt+spacing);
labelStr='Problem Type';
labelList='Problem1|Problem2';
callbackStr='equilibrium2DP eval';
% Generic label information
labelPos=[left yLabelPos-labelHt labelWid labelHt];
uicontrol(...
'Style','text', ...
'Units','normalized', ...
'Position',labelPos, ...
'BackgroundColor',labelColor, ...
'HorizontalAlignment','left', ...
'String',labelStr);
% Generic popup button information
btnPos=[left yLabelPos-labelHt-btnHt-btnOffset btnWid btnHt];
hndl1=uicontrol( ... 'Style','popup', ... 'Units','normalized', ... 'Position',btnPos, ... 'String',labelList, ... 'Callback',callbackStr);

% The FORCES command popup button
btnNumber=2; yLabelPos=top-(btnNumber-1)*(btnHt+labelHt+spacing); labelStr=' Change Forces'; labelList='As Given|Modify'; callbackStr='equilibrium2DP eval';

% Generic label information
labelPos=[left yLabelPos-labelHt labelWid labelHt]; uicontrol( ... 'Style','text', ... 'Units','normalized', ... 'Position',labelPos, ... 'BackgroundColor',labelColor, ... 'HorizontalAlignment','left', ... 'String',labelStr);

% Generic popup button information
btnPos=[left yLabelPos-labelHt-btnHt-btnOffset btnWid btnHt]; hndl2=uicontrol( ... 'Style','popup', ... 'Units','normalized', ... 'Position',btnPos, ... 'String',labelList, ... 'Callback',callbackStr);

% The solve button.
ucicontrol( ... 'Style','pushbutton', ... 'Units','normalized', ... 'Position',[left bottom+3*btnHt+3*spacing btnWid 2*btnHt], ... 'String','Solve', ... 'Callback','equilibrium2DP("solve")');

% The info button.
ucicontrol( ... 'Style','pushbutton', ... 'Units','normalized', ... 'Position',[left bottom+2*btnHt+spacing btnWid 2*btnHt], ... 'String','Info', ... 'Callback','equilibrium2DP("info")');

% The close button.
ucicontrol( ... 'Style','pushbutton', ... 'Units','normalized', ... 'Position',[left bottom btnWid 2*btnHt], ... 'String','Close', ... 'Callback','close(gcf)');

% Uncover the figure
hndlList=[mcwHndl hndl1 hndl2]; watchoff(oldFigNumber); set(figNumber,'Visible','on', ... 'UserData',hndlList); equilibrium2DP('eval');

elseif strcmp(action,'info'),
    helpwin(mfilename)
elseif strcmp(action,'eval'),
% Acquire Problem statement and image
hndlList=get(gcf,'UserData');
newHndl=hndlList(1);
newStrList=get(hndlList(2),'String');
newStrVal=get(hndlList(2),'Value');
Problem=deblank(newStrList(newStrVal,:));
newStrList=get(hndlList(3),'String');
newStrVal=get(hndlList(3),'Value');
Modify=deblank(newStrList(newStrVal,:));

%load Image
I=imread(strcat(Problem,Modify),'tiff');
%load Problem
P = textread(strcat(Problem, '.txt'), '%s','whitespace', '');
%Display Problem Text
set(mcwHndl,'String',P);
%Display Image and Co-ordinate Axis
if strcmp(Problem,'Problem1'),
    xform  = [1 0 0; 0 -1 0; -206 145 1]; %Origin transfer, image invert for displaying graphics
elseif strcmp(Problem,'Problem2'),
    xform  = [1 0 0; 0 -1 0; -284 132 1];
end
tform_translate = maketform('affine',xform);
[I xdata ydata] = imtransform(I, tform_translate);
cla;
imshow(I, 'XData',xdata,'YData', ydata);
axis on; axis xy;
showx; showy; drawnow;

%Acquire Data, Display Forces (As Given / Modified)
P = char(P);
if strcmp(Problem,'Problem1'),
syms t; s=1; %Problem specific variables and graphics scaling
    switch Modify
        case 'As Given'
            Force = deg2xy(([150),240,0,0]);
        case 'Modify'
            %Ask User to Modify the 240 N force
            h = helpdlg('Choose a point from the figure axis specifying end point of Force Vector','Point Selection');% Provide One Force by specifying
            End points of Vectors
            waitfor(h);
            [x,y] = ginput(1); % Provide One Force by specifying
            Force = [x*s,y*s,0,0];
        end
    Forces  = [Force;deg2xy((t),75,0,0)]; ...
    deg2xy((t+50),75,0,0)];
    angle = xy2deg(subs(Force));
    Angles = [angle(1); t; t+50];
elseif strcmp(Problem,'Problem2'),
syms tac tbc; s=1/20; %Problem specific variables and graphics scaling
    switch Modify
        case 'As Given'
            y = 0;
        case 'Modify'
            %Ask User to Modify the location of Cs vertical position
            h = helpdlg('Choose a point from the figure axis specifying "Y"location of Force Vector','Point Selection');% Provide One Force by specifying
            waitfor(h);
            [x,y] = ginput(1); % Provide One Force by specifying
            end
    angleac = 180-atand((8.5-(5+(y*s)))/12);
    anglebc = atand((9-(5+(y*s)))/7.5);
    Forces = [deg2xy([-90],396,0,y)];
    deg2xy([angleac,tac,0,0]); ...
    deg2xy([anglebc,tbc,0,0])];
    Angles = [-90; angleac; anglebc];
end
showvect(double(Forces(1,:)),s,'b');
axis tight; drawnow;
newStr = strcat('Forces=',char(Forces));
set(mcwHndl,'String',strvcat(P,'Forces are represented as [Xmag Ymag Xpos Ypos] in N',newStr));
set(hndlList(3), 'Userdata', struct('Forces',Forces,'Angles',Angles));

else if strcmp(action,'solve'),
  % Assemble and execute the completed command
  hndlList=get(gcf,'UserData');
  P = get(hndlList(1),'String');
  newStrList=get(hndlList(2),'String');
  newStrVal=get(hndlList(2),'Value');
  Problem=deblank(newStrList(newStrVal,:));
  Forces=get(hndlList(3),'Userdata');
  Angles = Forces.Angles; Forces = Forces.Forces;
  switch Problem
  case 'Problem1'
      t = findsym(Forces); s=1;
      Resultant = sumforce(Forces);
      sol = simplify(solve(Resultant(2)));
      alpha = double(sol);
      % find the correct angle
      check = double(subs(Resultant,t,alpha(1)));
      if check(1)>0
        alpha = alpha(2);
      else
        alpha = alpha(1);
      end
      Resultant = double(subs(Resultant,t,alpha));
      Forces = double(subs(Forces,t,alpha));
      case 'Problem2'
      [tac tbc] = strread(findsym(Forces), '%s %s', 'delimiter', ',');s =1/20;
      Forces(2:3,:) = twovector(subs(Forces(1,:)),[DR(Angles(2)) DR(Angles(3))]);
      Forces = double(subs(Forces))
  end
  showvect(Forces,s,'b');
  Forces = xy2deg(Forces);
  P = strvcat(P,'Forces are represented as [Ang Mag Xpos Ypos] in N',strcat('Forces=',mat2str(Forces)));
  if exist('Resultant','var'),
    showvect(Resultant,s,'r');
    Resultant = xy2deg(Resultant);
    P = strvcat(P,strcat('Resultant=',mat2str(Resultant)));
  end
  if strcmp(Problem, 'Problem1'), P = strvcat(P,strcat('Alpha=',mat2str(alpha)));
  end
  set(hndlList(1),'String',P); axis auto;
end;    % if strcmp(action, ...

A.2 equilibrium2DR.m Matlab GUI file

function equilibrium2DR(action)
  % equilibrium2DR is a GUI in Matlab to Demonstrate 2D equilibrium of
  % Rigid Bodies. This window allows the User to input a 2D equilibrium problem
  % (a text file describing the problem, coordinates, units and a figure file showing the
  % structure for the 2D equilibrium problem and provides the required solution.
  % By playing with the popup menus on the right side of the window, the User can
  % adjust the type of problem, the solutions required and so on.
  % The MiniCommand Window in the lower right shows the problem descriptions
  % and also solution values.
  % User m functions called: deg2xy.m, reaction.m and xy2deg.m
  % The user may, and should modify the M-files as they see fit.
  % These were developed using MATLAB 7.3.0.267 (R2006b). Although not
  % fully tested on older versions, they seem to be backwards
  % compatible.
  % DISCLAIMER:
  % "These M-Files are User Contributed Routines which are upon request
  % distributed on an "as is" basis. A User Contributed Routine is not a
if nargin<1,
    action='initialize';
end;

if strcmp(action,'initialize'),
    oldFigNumber=watchon;
    scrsz = get(0,'ScreenSize');
    figNumber=figure( ...  
        'Visible','off', ...  
        'NumberTitle','off', ...  
        'Color','black', ...  
        'Name','2-D Equilibrium of Rigid Bodies', ...  
        'Position',scrsz);
    colordef(figNumber,'none')
axes( ...  
        'Units','normalized', ...  
        'Position',[0.07 0.45 0.60 0.50]);

% Set up the Problem statement Window
    top=0.35;
    left=0.05;
    right=0.70;
    bottom=0.05;
    labelHt=0.05;
    spacing=0.005;
    % First, the Problem statement frame
    frmBorder=0.02;
    frmPos=[left-frmBorder bottom-frmBorder ... 
            (right-left)+2*frmBorder (top-bottom)+2*frmBorder];
    uicontrol( ...  
        'Style','frame', ...  
        'Units','normalized', ...  
        'Position',frmPos, ...  
        'BackgroundColor',[0.50 0.50 0.50]);
    % Then the text label
    labelPos=[left top-labelHt (right-left) labelHt];
    uicontrol( ...  
        'Style','text', ...  
        'Units','normalized', ...  
        'Position',labelPos, ...  
        'BackgroundColor',[0.50 0.50 0.50], ...  
        'ForegroundColor',[1 1 1], ...  
        'String','Problem Statement');
    % Then the editable text field
    mcwPos=[left bottom (right-left) top-bottom-labelHt-spacing];
    mcwHndl=uicontrol( ...  
        'Style','edit', ...  
        'HorizontalAlignment','left', ...  
        'Units','normalized', ...  
        'Max',10, ...  
        'BackgroundColor',[1 1 1], ...  
        'Position',mcwPos);
    % Save this handle for future use
    set(gcf,'UserData',mcwHndl);

% Information for all buttons
    labelColor=[0.8 0.8 0.8];
    top=0.95;
    bottom=0.05;
    left=0.75;
    yInitLabelPos=0.90;
left=0.75;
labelWid=0.20;
labelHt=0.05;
btnWid=0.20;
btnHt=0.05;
% Spacing between the label and the button for the same command
btnOffset=0.003;
% Spacing between the button and the next command's label
spacing=0.05;

% The CONSOLE frame
frmBorder=0.02;
yPos=0.05frmBorder;
frmPos=[left-frmBorder yPos btnWid+2*frmBorder 0.9+2*frmBorder];
h=uicontrol( ... 
'Style','frame', ... 
'Units','normalized', ... 
'Position',frmPos, ... 
'BackgroundColor',[0.50 0.50 0.50]);

% The PROBLEM INPUT command popup button
btnNumber=1;
yLabelPos=top-(btnNumber-1)*(btnHt+labelHt+spacing);
labelStr='Problem Type';
labelList='Problem1';% 'Problem1|Problem2|...
callbackStr='equilibrium2DR eval';

% Generic label information
labelPos=[left yLabelPos-labelHt labelWid labelHt];
uicontrol( ... 
'Style','text', ... 
'Units','normalized', ... 
'Position',labelPos, ... 
'BackgroundColor',labelColor, ... 
'HorizontalAlignment','left', ... 
'String',labelStr);

% Generic popup button information
btnPos=[left yLabelPos-labelHt-btnHt-btnOffset btnWid btnHt];
hndl1=uicontrol( ... 
'Style','popup', ... 
'Units','normalized', ... 
'Position',btnPos, ... 
'String',labelList, ... 
'Callback',callbackStr);

% The FORCES command popup button
btnNumber=2;
yLabelPos=top-(btnNumber-1)*(btnHt+labelHt+spacing);
labelStr='Change Forces';
labelList='As Given|Modify';
callbackStr='equilibrium2DR eval';

% Generic label information
labelPos=[left yLabelPos-labelHt labelWid labelHt];
uicontrol( ... 
'Style','text', ... 
'Units','normalized', ... 
'Position',labelPos, ... 
'BackgroundColor',labelColor, ... 
'HorizontalAlignment','left', ... 
'String',labelStr);

% Generic popup button information
btnPos=[left yLabelPos-labelHt-btnHt-btnOffset btnWid btnHt];
hndl2=uicontrol( ... 
'Style','popup', ...
% The solve button.
uicontrol( ... 'Style','pushbutton', ..., 'Units','normalized', ..., 'Position',[left bottom+3*btnHt+3*spacing btnWid 2*btnHt], ..., 'String','Solve', ..., 'Callback','equilibrium2DR("solve"));

% The info button.
uicontrol( ... 'Style','pushbutton', ..., 'Units','normalized', ..., 'Position',[left bottom+2*btnHt+spacing btnWid 2*btnHt], ..., 'String','Info', ..., 'Callback','equilibrium2DR("info"));

% The close button.
uicontrol( ... 'Style','pushbutton', ..., 'Units','normalized', ..., 'Position',[left bottom btnWid 2*btnHt], ..., 'String','Close', ..., 'Callback','close(gcf));

% Uncover the figure
hndlList=[mcwHndl hndl1 hndl2];
watchoff(oldFigNumber);
set(figNumber,'Visible','on', ...
'UserData',hndlList);
equilibrium2DR('eval');

elseif strcmp(action,'info'),
  helpwin(mfilename)
elseif strcmp(action,'eval'),
  % Acquire Problem statement and image
  hndlList=get(gcf,'UserData');
  mcwHndl=hndlList(1);
  newStrList=get(hndlList(2),'String');
  newStrVal=get(hndlList(2),'Value');
  Problem=deblank(newStrList(newStrVal,:));
  newStrList=get(hndlList(3),'String');
  newStrVal=get(hndlList(3),'Value');
  Modify=deblank(newStrList(newStrVal,:));
  %load Image
  I=imread(strcat(Problem,Modify),'tiff');
  %load Problem
  P = textread(strcat(Problem, '.txt'), '%s','whitespace', '');
  %Display Problem Text
  set(mcwHndl,'String',P);
  %Display Image and Co-ordinate Axis
  cla;
  imshow(I, 'XData',xdata,'YData', ydata);
  axis on; axis xy;
  showx; showy; drawnow;
end
%Acquire Data, Display Forces (As Given / Modified)
P = char(P);
if strcmp(Problem,'Problem1'),
syms t; s=1; %Problem specific variables and graphics scaling
switch Modify
  case 'As Given'
    Forces = deg2xy([210,500,5,0]);
    Moments = [3000,0,0,1]; %last column = 1 for Couple, 0 for Moment
  case 'Modify'
    %Ask User to Modify the force
    prompt={'Modify the Magnitude of 500N Force','Modify the Angle (deg) of 500N Force:','Modify the Position of 500N Force:','Modify the 3000Nm Couple:'};
    name='Modify the Force and Couple Vector';
    numlines=1;
    defaultanswer={'500','210','5','3000'};
    answer=inputdlg(prompt,name,numlines,defaultanswer);
    answer = char(answer);
    Forces =deg2xy([str2num(answer(2,:)),str2num(answer(1,:)),str2num(answer(3,:)), 0]);
    Moments = [str2num(answer(4,:)),0,0,1];
  end
elseif strcmp(Problem,'Problem2'),
%Suitable commands for Problem 2
end
axis tight;
ewStr1 = strcat('Forces=',mat2str(Forces));
ewStr2 = strcat('Moments=',mat2str(Moments));
set(mcwHndl,'String',strvcat(P,'Forces are represented as [Xmag Ymag Xpos Ypos] in N','Moments are represented as [mag Xpos Ypos (flag for Couple)] in Nm',newStr1,newStr2));
set(hndlList(3), 'Userdata', struct('Forces',Forces,'Moments',Moments));
elseif strcmp(action,'solve'),
% Assemble and execute the completed command
hndlList=get(gcf,'UserData');
P = get(hndlList(1),'String');
ewStrList=get(hndlList(2),'String');
ewStrVal=get(hndlList(2),'Value');
Problem=deblank(newStrList(newStrVal,:));
Forces=get(hndlList(3),{'Userdata',struct('Forces',Forces,'Moments',Moments)});
switch Problem
  case 'Problem1'
    [force,moment]=reaction(Forces,[0 0],Moments(:,1));
    Force = xy2deg(-force);
    Moment = -[moment,0,0,0];
  case 'Problem2'
    %Suitable commands for Problem 2
end
P = strvcat(P,'Resultant','Forces are represented as [Ang Mag Xpos Ypos] in N','Moments are represented as [mag Xpos Ypos (flag for Couple)] in Nm',strcat('Force=',mat2str(Force)),strcat('Moment=',mat2str(Moment)));
set(hndlList(1),'String',P); axis auto;
end;    % if strcmp(action, ...

A.3 Truss2D.m Matlab GUI file

function Truss2D(action)
% Truss2D is a GUI in Matlab to Demonstrate 2D equilibrium of Truss
% (Connected Rigid Bodies). This window allows the User to input a 2D Truss
% problem (a data text file describing the problem, coordinates, units and
% provides the required solution (Forces in Members, Support Reactions).
% By playing with the popup menus on the right side of the window, the User can
% adjust the type of problem, the solutions required and so on.
The MiniCommand Window in the lower right shows the problem descriptions and also solution values.

User m functions called: TP.m, ST.m and Problem1.m

The user may, and should modify the M-files as they see fit.

These were developed using MATLAB 7.3.0.267 (R2006b). Although not fully tested on older versions, they seem to be backwards compatible.

DISCLAIMER:
"These M-Files are User Contributed Routines which are upon request distributed on an "as is" basis. A User Contributed Routine is not a product of The Math Works, Inc. and The Math Works or the author assumes no responsibility for any errors that may exist in these routines."

Author : G Saravana Kumar, Year Created: 2008
All rights reserved with the Author.

if nargin<1,
    action='initialize';
end;

if strcmp(action,'initialize'),
    oldFigNumber=watchon;
    scrsz = get(0,'ScreenSize');
    figNumber=figure( ... 'Visible','off', ... 'NumberTitle','off', ... 'Color','black', ... 'Name','2-D Equilibrium of Connected Rigid Bodies (Truss)', ... 'Position',scrsz);
    colordef(figNumber,'none')
    axes( ... 'Units','normalized', ... 'Position',[0.07 0.45 0.60 0.50]);

% Set up the Problem statement Window

top=0.35;
left=0.05;
right=0.70;
bottom=0.05;
labelHt=0.05;
spacing=0.005;
% First, the Problem statement frame
frmBorder=0.02;
frmPos=[left-frmBorder bottom-frmBorder ... (right-left)+2*frmBorder (top-bottom)+2*frmBorder];
uicontrol( ... 'Style','frame', ... 'Units','normalized', ... 'Position',frmPos, ... 'BackgroundColor',[0.50 0.50 0.50]);
% Then the text label
labPos=[left bottom (right-left) top-bottom-labelHt-spacing];
uicontrol( ... 'Style','text', ... 'Units','normalized', ... 'Position',labPos, ... 'BackgroundColor',[1 1 1], ... 'ForegroundColor',[1 1 1], ... 'String','Problem Statement');
% Then the editable text field
mcwPos=[left bottom (right-left) top-bottom-labelHt-spacing];
mcwHndl=uicontrol(...
  'Style','edit',...
  'HorizontalAlignment','left',...
  'Units','normalized',...
  'Max',10,...
  'BackgroundColor',[1 1 1],...
  'Position',mcwPos);
%
% Save this handle for future use
set(gcf,'UserData',mcwHndl);
%
%================================================================================
%
% Information for all buttons
labelColor=[0.8 0.8 0.8];
top=0.95;
botton=0.05;
left=0.75;
yInitLabelPos=0.90;
left=0.75;
labelWid=0.20;
labelHt=0.05;
btnWid=0.20;
btnHt=0.05;
%
% Spacing between the label and the button for the same command
btnOffset=0.003;
%
% Spacing between the button and the next command's label
spacing=0.05;
%
%================================================================================
%
% The CONSOLE frame
frmBorder=0.02;
yPos=0.05-frmBorder;
frmPos=[left-frmBorder yPos btnWid+2*frmBorder 0.9+2*frmBorder];
h=uicontrol(...
  'Style','frame',...
  'Units','normalized',...
  'Position',frmPos,..
  'BackgroundColor',[0.50 0.50 0.50]);
%
%================================================================================
%
% The PROBLEM INPUT command popup button
btnNumber=1;
yLabelPos=top-(btnNumber-1)*(btnHt+labelHt+spacing);
labelStr=' Problem Type';
labelList='Problem1';% 'Problem1|Problem2|...' 
callbackStr='Truss2D eval';
%
% Generic label information
labelPos=[left yLabelPos-labelHt labelWid labelHt];
uicontrol(...
  'Style','text',...
  'Units','normalized',...
  'Position',labelPos, ...
  'BackgroundColor',labelColor, ...
  'HorizontalAlignment','left', ...,
  'String',labelStr);
%
% Generic popup button information
btnPos=[left yLabelPos-labelHt-btnHt-btnOffset btnWid btnHt];
hndl1=uicontrol(...
  'Style','popup',...
  'Units','normalized', ...
  'Position',btnPos, ...
  'BackgroundColor',labelColor, ...
  'HorizontalAligment','left', ...,
  'String',callbackStr);
%
%================================================================================
%
% The FORCES command popup button
btnNumber=2;
yLabelPos=top-(btnNumber-1)*(btnHt+labelHt+spacing);
labelStr=' Change Truss Geometry, Reactions and Forces';
labelList='As Given|Modify';
callbackStr='Truss2D eval';

% Generic label information
labelPos=[left yLabelPos-labelHt labelWid labelHt];
uicontrol( ...
 'Style','text', ...
 'Units','normalized', ...
 'Position',labelPos, ...
 'BackgroundColor',labelColor, ...
 'HorizontalAlignment','left', ...
 'String',labelStr);

% Generic popup button information
btnPos=[left yLabelPos-labelHt-btnHt-btnOffset btnWid btnHt];
hndl2=uicontrol( ...
 'Style','popup', ...
 'Units','normalized', ...
 'Position',btnPos, ...
 'String',labelList, ...
 'Callback',callbackStr);

%====================================
% The solve button.
uicontrol( ...
 'Style','pushbutton', ...
 'Units','normalized', ...
 'Position',[left bottom+3*btnHt+3*spacing btnWid 2*btnHt], ...
 'String','Solve', ...
 'Callback','Truss2D("solve")');

% The info button.
uicontrol( ...
 'Style','pushbutton', ...
 'Units','normalized', ...
 'Position',[left bottom+2*btnHt+spacing btnWid 2*btnHt], ...
 'String','Info', ...
 'Callback','Truss2D("info")');

% The close button.
uicontrol( ...
 'Style','pushbutton', ...
 'Units','normalized', ...
 'Position',[left bottom btnWid 2*btnHt], ...
 'String','Close', ...
 'Callback','close(gcf)');

% Uncover the figure
hndlList=[mcwHndl hndl1 hndl2];
watchoff(oldFigNumber);
set(figNumber,'Visible','on', ...
 'UserData',hndlList);
Truss2D('eval');

elseif strcmp(action,'info'),
    helpwin(mfilename)
elseif strcmp(action,'eval'),
% Acquire Problem statement and image
hndlList=get(gcf,'UserData');
mcwHndl=hnndlList(1);
newStrList=strcmp(hndlList(2),'String');
newStrVal=strcmp(hndlList(3),'Value');
Problem=deblank(newStrList(newStrVal,:));
newStrList=strcmp(hndlList(3),'Value');
Modify=deblank(newStrList(newStrVal,:));

% Load Problem
Problem1.m Matlab Input file

function D=Problem1(),
% Definition of Data
% Nodal Coordinates
Coord=360*[2 1 0 0; 1 1 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 0];
% Connectivity
Con=[5 3 1 3 6 4 2 4 3 2 1 6 3 5 4 4 1 3 2];
% Definition of Degree of freedom (free=0 & fixed=1); for 2-D trusses the last column is equal to 1
Re=[0 0 1 0 0 0 1 0 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0];
% Definition of Nodal loads
Load=zeros(size(Coord));Load(2,:)=[0 -1e5 0];Load(4,:)=[0 -1e5 0];
% Definition of Modulus of Elasticity
E=ones(1,size(Con,1))*1e30;
% Definition of Area

% Problem specific variables and graphics scaling
switch Modify
    case 'As Given'
    clear Problem1;D = Problem1;
    Forces = [D.Load(1:2,:)' D.Coord(1:2,:)'];
    case 'Modify'
    % Ask User to Modify the Function Problem1
    edit Problem1.m;
    h = helpdlg('Edit the M file containing Data, save and then press OK','Modify Data');
    % Provide One Force by specifying End points of Vectors
    waitfor(h);
    clear Problem1;D = Problem1;
    Forces = [D.Load(1:2,:)' D.Coord(1:2,:)'];
end
% Display Image and Co-ordinate Axis
TP(D);
drawnow;
else if strcmp(Problem,'Problem2'),
% Suitable commands for Problem 2
end
newStr1 = strcat('Coordinates of Nodes=',mat2str(D.Coord));
newStr2 = strcat('Members=',mat2str(D.Con));
newStr3 = strcat('Degrees of Freedom (X,Y,Z and 0 - free, 1- fixed,mat2str(D.Re));
newStr4 = strcat('Applied Forces=',mat2str(Forces));
set(newHndl,'String',strvcat(P,newStr1,newStr2,newStr3,'Forces are represented as [Xmag Ymag Xpos Ypos] in N',newStr4));
set(hndlList(3), 'Userdata', D);
elseif strcmp(action,'solve'),
% Assemble and execute the completed command
hndlList=get(gcf,'UserData');
P = get(hndlList(1),'String');
newStrList=get(hndlList(2),'String');
newStrVal=get(hndlList(2),'Value');
Problem=deblank(newStrList(newStrVal,:));
D=get(hndlList(3),'UserData');
switch Problem
    case 'Problem1'
    [F,U,R]=ST(D);
    case 'Problem2'
% Suitable commands for Problem 2
end
P = strvcat(P,'Forces (N) in Members 1 2...n',strcat('Force=',mat2str(F)),'Reaction(N) at Supports (Matrix with n*3 (n = No. of Joints)),strcat('Reaction=',mat2str(R')));
set(hndlList(1),'String',P); axis auto;
end;    % if strcmp(action, ...
A=[27.5 0.1 24.5 17.5 0.1 0.5 7.5 21.5 21.5 0.1];
% Convert to structure array
D=struct('Coord',Coord,'Con',Con,'Re',Re,'Load',Load,'E',E,'A',A);
return;

A.4 Beams.m Matlab GUI file

function Beams(action)
  % Beams is a GUI in MAtlab that Demonstrate 2D equilibrium of User Defined
  % Beams. This window allows the user to input a 2D Beam problem (a data text
  % file describing the beam, end conditions, loading and provides the required
  % solution (Reactions, Shear Force Diagram, Bending Moment Diagram).
  %
  % By playing with the popup menus on the right side of the window, the User can
  % adjust the type of problem, the solutions required and so on.
  %
  % The MiniCommand Window in the lower right shows the problem descriptions
  % and also solution values.
  %
  % User m functions called: TB.m, plotSMD.m and Problem1.m
  %
  % The user may, and should modify the M-files as they see fit.
  %
  % These were developed using MATLAB 7.3.0.267 (R2006b). Although not
  % fully tested on older versions, they seem to be backwards
  % compatible.
  %
  % DISCLAIMER:
  % "These M-Files are User Contributed Routines which are upon request
  % distributed on an "as is" basis. A User Contributed Routine is not a
  % product of The Math Works, Inc. and The Math Works or the author assumes
  % no responsibility for any errors that may exist in these routines."
  %
  % Author : G Saravana Kumar, Year Created: 2008
  % All rights reserved with the Author.
  %-----------------------------------------------------------------------
  if nargin<1,
    action='initialize';
  end;
  
  if strcmp(action,'initialize'),
    oldFigNumber=watchon;
    scrsz = get(0,'ScreenSize');
    figNumber=figure( ...
      'Visible','off', ...
      'NumberTitle','off', ...
      'Color','black', ...
      'Name','Beams', ...
      'Position',scrsz);
    colordef(figNumber,'none');
    AH = axes( ...
      'Units','normalized', ...
      'Position',[0.07 0.45 0.60 0.50]);
  
  % Set up the Problem statement Window
  top=0.35;
  left=0.05;
  right=0.70;
  bottom=0.05;
  labelHt=0.05;
  spacing=0.005;
  % First, the Problem statement frame
  frmBorder=0.02;
  frmPos=[left-frmBorder bottom-frmBorder ... (right-left)+2*frmBorder (top-bottom)+2*frmBorder];
  uicontrol( ...
'Style','frame', ...
'Units','normalized', ...
'Position',frmPos, ...
'BackgroundColor',[0.50 0.50 0.50]);

% Then the text label
labelPos=[left top-labelHt (right-left) labelHt];
uicontrol( ... 
'Style','text', ...
'Units','normalized', ...
'Position',labelPos, ...
'BackgroundColor',[0.50 0.50 0.50], ...
'ForegroundColor',[1 1 1], ...
'String','Problem Statement');

% Then the editable text field
cmpPos=[left bottom (right-left) top-bottom-labelHt-spacing];
mcwHndl=uicontrol( ... 
'Style','edit', ...
'HorizontalAlignment','left', ...
'Units','normalized', ...
'Max',10, ...
'BackgroundColor',[1 1 1], ...
'Position',mcwPos);

% Save this handle for future use
set(gcf,'UserData',mcwHndl);

%====================================
% Information for all buttons
labelColor=[0.8 0.8 0.8];
top=0.95;
bottom=0.05;
left=0.75;
yInitLabelPos=0.90;
left=0.75;
labelWid=0.20;
labelHt=0.05;
btnWid=0.20;
btnHt=0.05;

% Spacing between the label and the button for the same command
btnOffset=0.003;

% Spacing between the button and the next command's label
spacing=0.05;

%====================================
% The CONSOLE frame
frmBorder=0.02;
yPos=0.05-frmBorder;
frmPos=[left-frmBorder yPos btnWid+2*frmBorder 0.9+2*frmBorder];
h=uicontrol( ... 
'Style','frame', ...
'Units','normalized', ...
'Position',frmPos, ...
'BackgroundColor',[0.50 0.50 0.50]);

%====================================
% The PROBLEM INPUT command popup button
btnNumber=1;
yLabelPos=top-(btnNumber-1)*(btnHt+labelHt+spacing);
labelStr='Problem Type';
labelList='Problem1';% 'Problem1|Problem2|...'
callbackStr='Beams eval';

% Generic label information
labelPos=[left yLabelPos-labelHt labelWid+btnHt+2*frmBorder 0.9+2*frmBorder];
uicontrol( ... 
'Style','text', ...
'Units','normalized', ...
'Position',labelPos, ...
'BackgroundColor',labelColor, ...
'HorizontalAlignment','left', ...
'String',labelStr);
% Generic popup button information
btnPos=[left yLabelPos-labelHt-btnHt-btnOffset btnWid btnHt];
hndl1=uicontrol( ... 'Style','popup', ... 'Units','normalized', ... 'Position',btnPos, ... 'String',labelList, ... 'Callback',callbackStr);

%====================================
% The FORCES command popup button
btnNumber=2;
yLabelPos=top-(btnNumber-1)*(btnHt+labelHt+spacing);
labelStr=' Change Beam length, Forces, Moments and Reactions';
labelList='As Given|Modify';
callbackStr='Beams eval';

% Generic label information
labelPos=[left yLabelPos-labelHt labelWid labelHt];
uicontrol( ... 'Style','text', ... 'Units','normalized', ... 'Position',labelPos, ... 'BackgroundColor',labelColor, ... 'HorizontalAlignment','left', ... 'String',labelStr);

% Generic popup button information
btnPos=[left yLabelPos-labelHt-btnHt-btnOffset btnWid btnHt];
hndl2=uicontrol( ... 'Style','popup', ... 'Units','normalized', ... 'Position',btnPos, ... 'String',labelList, ... 'Callback',callbackStr);

%====================================
% The solve button.
uicontrol( ... 'Style','pushbutton', ... 'Units','normalized', ... 'Position',[left bottom+3*btnHt+3*spacing btnWid 2*btnHt], ... 'String','Solve', ... 'Callback','Beams(''solve''));

% The info button.
uicontrol( ... 'Style','pushbutton', ... 'Units','normalized', ... 'Position',[left bottom+2*btnHt+spacing btnWid 2*btnHt], ... 'String','Info', ... 'Callback','Beams(''info''));

% The close button.
uicontrol( ... 'Style','pushbutton', ... 'Units','normalized', ... 'Position',[left bottom btnWid 2*btnHt], ... 'String','Close', ... 'Callback','close(gcf)');

% Uncover the figure
hndlList=[mcwHndl hndl1 hndl2 AH];
watchoff(oldFigNumber);
set(figNumber,'Visible','on', ... 'UserData',hndlList);
Beams('eval');

elseif strcmp(action,'info'),
helpwin(mfilename)

elseif strcmp(action,'eval'),
% Acquire Problem statement and image
hndlList=get(gcf,'UserData');
cmcWndl=hndlList(1);
newStrList=get(hndlList(2),'String');
newStrVal=get(hndlList(2),'Value');
Problem=deblank(newStrList(newStrVal,:));
newStrList=get(hndlList(3),'String');
newStrVal=get(hndlList(3),'Value');
Modify=deblank(newStrList(newStrVal,:));
AH = hndlList(4);
%load Problem
P = textread(strcat(Problem, '.txt'), '%s','whitespace', '');
%Display Problem Text
set(mcwHndl,'String',P);
%Acquire Data, Display Forces (As Given / Modified)
P = char(P);
if strcmp(Problem,'Problem1'),
s=1; %Problem specific variables and graphics scaling
    switch Modify
        case 'As Given'
            clear Problem1;D = Problem1;
        case 'Modify'
            %Ask User to Modify the Function Problem1
            edit Problem1.m;
            h = helpdlg('Edit the M file containing Data, save and then press OK', 'Modify Data');% Provide One Force by specifying End points of Vectors
            waitfor(h);
            clear Problem1;D = Problem1;
    end
    %Display Image and Co-ordinate Axis
    TB(D,AH);
    elseif strcmp(Problem,'Problem2'),
%Suitable commands for Problem 2
    end
end
newStr1 = strcat('Beam Span=',mat2str(D.Length));
newStr2 = strcat('Reaction Unknown=',mat2str(D.Unknowns));
newStr3 = strcat('Applied Forces=',mat2str(D.Force));
newStr4 = strcat('Applied Moment if any=',mat2str(D.MMoments));
newStr5 = strcat('Moment Location if any=',mat2str(D.MLocation));
newStr6 = strcat('Distributed Force if any=',mat2str(D.DForces));
set(mcwHndl,'String',strvcat(P,newStr1,newStr2,newStr3,newStr4,newStr5,newStr6, ...
'Forces are represented as [Xmag Ymag Xpos Ypos] in N', ...
'Moments in Nm','Unknown reactions represented as direction and location', ...
'Distributed Force as [Mag 1 Mag2 span location]'));
set(hndlList(3), 'Userdata', D);
elseif strcmp(action,'solve'),
% Assemble and execute the completed command
hndlList=get(gcf,'UserData');
P = get(hndlList(1),'String');
ewStrList=get(hndlList(2),'String');
newStrVal=get(hndlList(2),'Value');
Problem=deblank(newStrList(newStrVal,:));
D=get(hndlList(3),'Userdata');
AH = hndlList(4);
switch Problem
    case 'Problem1'
        Reactions=threevector(D.Force,D.Unknowns,-D.MMoments);
        x=0:0.1:D.Length;
        % Include Loads in the order of appearence
        s(1,:)=diagram(x,'point',D.Force(1,2),D.Force(1,3)); % point loads
        s(2,:)=diagram(x,'point',Reactions(1,2),Reactions(1,3)); % Reaction
        s(3,:)=diagram(x,'distributed',D.DForces(1,2),D.DForces(1,3)); % Distributed Load
        s(4,:)=diagram(x,'point',Reactions(2,2),Reactions(2,3)); % Reaction
        Shear=sum(s);
m(1,:)=diagram(x,'point',D.Moments(1),D.MLocation(1));
m(2,:)=diagramintegral(x,Shear);
Moment=sum(m);
interpolate (x,Shear,12.33);
MaxShear=max(abs(Shear));
MaxMoment=max(abs(Moment));
XatMaxMoment=x(find(abs(Moment)==max(abs(Moment))));
XatMaxShear=x(find(abs(Shear)==max(abs(Shear))));

case 'Problem2'
  %Suitable commands for Problem 2
end

P = strvcat(P,strcat('Reactions=',mat2str(Reactions)),...
        strcat('MaxShear(N)=',mat2str(MaxShear)),...
        strcat('MaxMoment(Nm)=',mat2str(MaxMoment)), ...
        strcat('MaxMoment Location (from right) in m =',mat2str(XatMaxMoment)));
set(hndlList(1),'String',P); axis auto;
end;  % if strcmp(action, ...

Problem1.m Matlab Input file

function D=Problem1(),
% Definition of Data
%Length of the Beam
L=20;
%Types of Reactions (3 with known direction and location)
Unknowns=[DR(90) 5 0; DR(90) 19 0;0 5 0];
% Acting Couple Moment
Moment=30; % Could be a list as Vector
% Location of Moment
ML = 0; % Could be a list as vector
%Acting Force 1 (concentrated Force)
af(1,:)=[0 -10 0 0];
%Acting Force 2 (Distributed Force)
%Distributed Load (convert to equivalent Point load
[DLForce DLPlacement]=distload(-30,-30,3);
af(2,:)=[0 DLForce 10+DLPlacement 0];
D=struct('Length',L,'Unknowns',Unknowns,'Moments',Moment,'MLocation',ML,'Force',af,'DForces',[-30 -30 3 10]);
%Linearly Varying DForce = [Mag1 Mag2 span Location]
return;

A.5 Friction.m Matlab GUI file

function Friction(action)
% Friction is a GUI in Matlab to Demonstrate the effect of Co-efficeint of Friction
% on the equilibrium of Rigid Bodies. This window allows the User to input a Problem
% (image as well as data) and obtain results (such as the range of values of equilibrium
% position for a range of Co-efficeint of Friction).
% By playing with the popup menus on the right side of the window, the User can
% adjust the type of problem, the solutions required and so on.
% The MiniCommand Window in the lower right shows the problem descriptions
% and also solution values.
% User m functions called: Problem1.m
%
% These were developed using MATLAB 7.3.0.267 (R2006b). Although not
% fully tested on older versions, they seem to be backwards
% compatible.

% DISCLAIMER:
% "These M-Files are User Contributed Routines which are upon request
if nargin<1,
    action='initialize';
end;

if strcmp(action,'initialize'),
    oldFigNumber=watchon;
    scrsz = get(0,'ScreenSize');
    figNumber=figure( ...
        'Visible','off', ... 
        'NumberTitle','off', ... 
        'Color','black', ... 
        'Name','Friction', ... 
        'Position',scrsz);
    colordef(figNumber,'none');
    AH = axes( ...
        'Units','normalized', ... 
        'Position',[0.07 0.45 0.60 0.50]);

    %===================================
    % Set up the Problem statement Window
    top=0.35;
    left=0.05;
    right=0.70;
    bottom=0.05;
    labelHt=0.05;
    spacing=0.005;
    % First, the Problem statement frame
    frmBorder=0.02;
    frmPos=[left-frmBorder bottom-frmBorder ...
            (right-left)+2*frmBorder (top-bottom)+2*frmBorder];
    uicontrol( ...
        'Style','frame', ... 
        'Units','normalized', ... 
        'Position',frmPos, ...
        'BackgroundColor',[0.50 0.50 0.50]);
    % Then the text label
    labelPos=[left top-labelHt (right-left) labelHt];
    uicontrol( ...
        'Style','text', ... 
        'Units','normalized', ... 
        'Position',labelPos, ...
        'BackgroundColor',[0.50 0.50 0.50]);
    % Then the editable text field
    mcwPos=[left bottom (right-left) top-bottom-labelHt-spacing];
    mcwHndl=uicontrol( ...
        'Style','edit', ... 
        'HorizontalAlignment','left', ... 
        'Units','normalized', ... 
        'Max',10, ...
        'BackgroundColor',[1 1 1], ...
        'Position',mcwPos);
    % Save this handle for future use
    set(gcf,'UserData',mcwHndl);

    %====================================
    % Information for all buttons
    labelColor=[0.8 0.8 0.8];
    top=0.95;
    bottom=0.05;
left=0.75;
yInitLabelPos=0.90;
left=0.75;
labelWid=0.20;
labelHt=0.05;
btnWid=0.20;
btnHt=0.05;
% Spacing between the label and the button for the same command
btnOffset=0.003;
% Spacing between the button and the next command's label
spacing=0.05;

% The CONSOLE frame
frmBorder=0.02;
yPos=0.05-frmBorder;
frmPos=[left-frmBorder yPos btnWid+3*frmBorder 0.9+2*frmBorder];
h=uicontrol( ... 
'Style','frame', ... 
'Units','normalized', 
'Position',frmPos, 
'BackgroundColor',[0.50 0.50 0.50]);

% The PROBLEM INPUT command popup button
btnNumber=1;
yLabelPos=top-(btnNumber-1)*(btnHt+labelHt+spacing);
labelStr='Problem Type';
labelList='Problem1';% 'Problem1|Problem2|...' 
callbackStr='Friction eval';

% Generic label information
labelPos=[left yLabelPos-labelHt labelWid labelHt];
uicontrol( ... 
'Style','text', ... 
'Units','normalized', ... 
'Position',leftPos, ... 
'BackgroundColor',labelColor, ... 
'HorizontalAlignment','left', ... 
'String',labelStr);

% Generic popup button information
btnPos=[left yLabelPos-labelHt btnHt btnHt-btnOffset btnWid btnHt];
hndl1=uicontrol( ... 
'Style','popup', ... 
'Units','normalized', ... 
'Position',btnPos, ... 
'String',labelList, ... 
'Callback',callbackStr);

% The FORCES command popup button
btnNumber=2;
yLabelPos=top-(btnNumber-1)*(btnHt+labelHt+spacing);
labelStr='Change Data';
labelList='As Given|Modify';
callbackStr='Friction eval';

% Generic label information
labelPos=[left yLabelPos-labelHt labelWid labelHt];
uicontrol( ... 
'Style','text', ... 
'Units','normalized', ... 
'Position',leftPos, ... 
'BackgroundColor',labelColor, ... 
'HorizontalAlignment','left', ... 
'String',labelStr);

% Generic popup button information
btnPos=[left yLabelPos-labelHt btnHt btnHt-btnOffset btnWid btnHt];
hndl2=uicontrol( ... 
'Style','popup', ... 
'Units','normalized', ... 
'Position',btnPos, ... 
'String',labelList, ... 
'Callback',callbackStr);

%====================================
% The solve button.
uicontrol( ... 
'Style','pushbutton', ... 
'Units','normalized', ... 
'Position',[left bottom+3*btnHt+3*spacing btnWid 2*btnHt], ... 
'String','Solve', ... 
'Callback','Friction("solve")');

% The info button.
uicontrol ... 
'Style','pushbutton', ... 
'Units','normalized', ... 
'Position',[left bottom+2*btnHt+spacing btnWid 2*btnHt], ... 
'String','Info', ... 
'Callback','Friction("info")');

% The close button.
uicontrol( ... 
'Style','pushbutton', ... 
'Units','normalized', ... 
'Position',[left bottom btnWid 2*btnHt], ... 
'String','Close', ... 
'Callback','close(gcf)');

% Uncover the figure
hndlList=[mcwHndl hndl1 hndl2 AH];
watchoff(oldFigNumber);
set(figNumber,'Visible','on', ... 
'UserData',hndlList);
Friction('eval');

elseif strcmp(action,'info'),
  helpwin(mfilename)
elseif strcmp(action,'eval'),
  % Acquire Problem statement and image
  hndlList=get(gcf,'UserData');
  mcwHndl=hndlList(1);
  newStrList=get(hndlList(2),'String');
  newStrVal=get(hndlList(2),'Value');
  Problem=deblank(newStrList(newStrVal,:));
  newStrList=get(hndlList(3),'String');
  newStrVal=get(hndlList(3),'Value');
  Modify=deblank(newStrList(newStrVal,:));
  AH = hndlList(4);
  %load Image
  I=imread(strcat(Problem,Modify),'tiff');
  %load Problem
  P = textread(strcat(Problem, '.txt'), '%s','whitespace', '');
  %Display Problem Text
  set(mcwHndl,'String',P);
  %Display Image
  axes(AH);
  imshow(I);
 axis off;

  P = char(P);
  %Acquire Data for the Problem
  if strcmp(Problem,'Problem1'),
    s=1; %Problem specific variables and graphics scaling
    switch Modify
    ...
case 'As Given'
clear Problem1;
D = Problem1;
case 'Modify'
% Ask User to Modify the Function Problem1
edit Problem1.m;
h = helpdlg('Edit the M file containing Data, save and then press OK','Modify Data');
% Provide One Force by specifying End points of Vectors
waitfor(h);
clear Problem1;
D = Problem1;
end
elseif strcmp(Problem,'Problem2'),
% Suitable commands for Problem 2
end
newStr1 = strcat('Masses m1 and m2 in kg=',mat2str([D.m1 D.m2]));
newStr2 = strcat('Range of Co-efficient of Friction=',mat2str(D.u));
set(mcwHndl,'String',strvcat(P,newStr1,newStr2));
set(hndlList(3), 'Userdata', D);
elseif strcmp(action,'solve'),
% Assemble and execute the completed command
hndlList=get(gcf,'UserData');
P = get(hndlList(1),'String');
newStrList=get(hndlList(2),'String');
ewStrVal=get(hndlList(2),'Value');
Problem=deblank(newStrList(newStrVal,:));
D=get(hndlList(3),'Userdata');
AH = hndlList(4);
switch Problem
  case 'Problem1'
syms u theta;
  Deltau = 0.05;
f = u*D.m1 + u*D.m2*sin(theta)-D.m2*cos(theta);
  Sol = solve(f,theta);
  THETA = [];
  for i=D.u(1):Deltau:D.u(2)
    THETA = [THETA subs(Sol,i)];
  end
  axes(AH);
  plot([D.u(1):Deltau:D.u(2)],THETA*180/pi,'.-y');
  grid on;
  xlabel('Coefficient of Friction'); ylabel('Angle Theta in Degrees');
  axis auto;
  case 'Problem2'
  % Suitable commands for Problem 2
  end
  P = strcat(P,'The Positive values for theta are for the intended motion to right', ...
  'The Negative values for theta are for the intended motion to left (consider the mirror of the Block Placement)');
  set(hndlList(1),'String',P);
end;  % if strcmp(action, ...

Problem1.m Matlab Input file

function D=Problem1(),
% Definition of Data for friction Problem1
m1 = 45; % Mass of Block 1 that impedes to slide
m2 = 36; % Mass of Block 2 that hangs
u = [0 0.6]; % Range of Co-efficient of Friction
% Convert to structure array
D=struct('m1',m1, 'm2',m2, 'u',u);
return;
A.6 Virtualwork.m Matlab GUI file

function Virtualwork(action)
    % Virtualwork is a GUI in Matlab to Demonstrate the use of the principle of Virtual Work in analyzing equilibrium of system of Rigid Bodies.
    % This window allows the User to input a Problem (image as well as data as a data file) and provides results like range of equilibrium position for combination of forces and position of a system.
    % By playing with the popup menus on the right side of the window, the User can adjust the type of problem, the solutions required and so on.
    % The MiniCommand Window in the lower right shows the problem descriptions and also solution values.
    % User m functions called: Problem1.m
    % The user may, and should modify the M-files as they see fit.
    % These were developed using MATLAB 7.3.0.267 (R2006b). Although not fully tested on older versions, they seem to be backwards compatible.
    % DISCLAIMER:
    % "These M-Files are User Contributed Routines which are upon request distributed on an "as is" basis. A User Contributed Routine is not a product of The Math Works, Inc. and The Math Works or the author assumes no responsibility for any errors that may exist in these routines."
    % Author : G Saravana Kumar, Year Created: 2008
    % All rights reserved with the Author.
    %-----------------------------------------------------------------------

if nargin<1,
   action='initialize';
end;

if strcmp(action,'initialize'),
   oldFigNumber=watchon;
   scrsz = get(0,'ScreenSize');
   figNumber=figure( ...
      'Visible','off', ...
      'NumberTitle','off', ...
      'Color','black', ...
      'Name','Virtualwork', ...
      'Position',scrsz);
   colordef(figNumber,'none');
   AH = axes( ...
      'Units','normalized', ...
      'Position','[0.07 0.45 0.60 0.50]');
   %=====================================================================
   % Set up the Problem statement Window
   top=0.35;
   left=0.05;
   right=0.70;
   bottom=0.05;
   labelHt=0.05;
   spacing=0.005;
   % First, the Problem statement frame
   frmBorder=0.02;
   frmPos=[left-frmBorder bottom-frmBorder ...
      (right-left)+2*frmBorder (top-bottom)+2*frmBorder];
   uicontrol( ...
'Units','normalized', ...
'Position','frmPos', ...
'BackgroundColor',[0.50 0.50 0.50]);
% Then the text label
labelPos=[left top LabelHt (right-left) labelHt];
uicontrol ...
 'Style','text', ...
'Units','normalized', ...
'Position','labelPos', ...
'BackgroundColor',[0.50 0.50 0.50], ...
'ForegroundColor',[1 1 1], ...
'String','Problem Statement');
% Then the editable text field
mcwPos=[left bottom (right-left) top-bottom-labelHt-spacing];
mcwHndl=uicontrol( ...
 'Style','edit', ...
'HorizontalAlignment','left', ...
'Units','normalized', ...
'Max',10, ...
'BackgroundColor',[1 1 1], ...
'Position',mcwPos);
% Save this handle for future use
set(gcf,'UserData',mcwHndl);
%===================================================================
% Information for all buttons
labelColor=[0.8 0.8 0.8];
top=0.95;
bottom=0.05;
left=0.75;
yInitLabelPos=0.90;
left=0.75;
labelWid=0.20;
labelHt=0.05;
btnWid=0.20;
btnHt=0.05;
% Spacing between the label and the button for the same command
btnOffset=0.003;
% Spacing between the button and the next command's label
spacing=0.05;
%===================================================================
% The CONSOLE frame
frmBorder=0.02;
yPos=0.05-frmBorder;
frmPos=[left-frmBorder yPos btnWid+2*frmBorder 0.9+2*frmBorder];
 h=uicontrol( ...
 'Style','frame', ...
'Units','normalized', ...
'Position',frmPos, ...
'BackgroundColor',[0.50 0.50 0.50]);
%===================================================================
% The PROBLEM INPUT command popup button
btnNumber=1;
yLabelPos=top-(btnNumber-1)*(btnHt+labelHt+spacing);
labelStr='Problem Type';
labelList='Problem1';% 'Problem1|Problem2|...'% Generic label information
labelPos=[left yLabelPos-labelHt labelWid labelHt];
uicontrol ...
 'Style','text', ...
'Units','normalized', ...
'Position','labelPos', ...
'BackgroundColor',labelColor, ...
'HorizontalAlignment','left', ...
'String',labelStr);
% Generic popup button information
btnPos=[left yLabelPos-labelHt-btnHt-btnOffset btnWid btnHt];
hndl1=uicontrol( ... 
'Style','popup', ...
'Units','normalized', ...
'Position',btnPos, ...
'String',labelList, ...
'Callback',callbackStr);

% The FORCES command popup button
btnNumber=2;
yLabelPos=top-(btnNumber-1)*(btnHt+labelHt+spacing);
labelStr=' Change Data';
labelList='As Given|Modify';
callbackStr='Virtualwork eval';

% Generic label information
labelPos=[left yLabelPos-labelHt labelWid labelHt];
uicontrol( ... 
'Style','text', ...
'Units','normalized', ...
'Position',labelPos, ...
'BackgroundColor',labelColor, ...
'HorizontalAlignment','left', ...
'String',labelStr);

% Generic popup button information
btnPos=[left yLabelPos-labelHt-btnHt-btnOffset btnWid btnHt];
hndl2=uicontrol( ... 
'Style','popup', ...
'Units','normalized', ...
'Position',btnPos, ...
'String',labelList, ...
'Callback',callbackStr);

% The solve button.
uicontrol( ... 
'Style','pushbutton', ...
'Units','normalized', ...
'Position',[left bottom+3*btnHt+3*spacing btnWid 2*btnHt], ...
'String','Solve', ... 
'Callback','Virtualwork("solve")');

% The info button.
uicontrol( ... 
'Style','pushbutton', ...
'Units','normalized', ...
'Position',[left bottom+2*btnHt+spacing btnWid 2*btnHt], ...
'String','Info', ... 
'Callback','Virtualwork("info")');

% The close button.
uicontrol( ... 
'Style','pushbutton', ...
'Units','normalized', ...
'Position',[left bottom btnWid 2*btnHt], ...
'String','Close', ... 
'Callback','close(gcf)');

% Uncover the figure
hndlList=[mcwHndl hndl1 hndl2 AH];
watchoff(oldFigNumber);
set(figNumber,'Visible','on', ... 
'UserData',hndlList);
Virtualwork('eval');

elseif strcmp(action,'info'),
helpwin(mfilename)
elseif strcmp(action,'eval'),
\% Acquire Problem statement and image
hndlList=get(gcf,'UserData');
mcwHndl=hndlList(1);
newStrList=get(hndlList(2),'String');
newStrVal=get(hndlList(2),'Value');
Problem=deblank(newStrList(newStrVal,:));
newStrList=get(hndlList(3),'String');
newStrVal=get(hndlList(3),'Value');
Modify=deblank(newStrList(newStrVal,:));
AH = hndlList(4);
\%load Image
I=imread(strcat(Problem,Modify),'tiff');
\%load Problem
P = textread(strcat(Problem, '.txt'), '%s','whitespace', '');
\%Display Problem Text
set(mcwHndl,'String',P);
\%Display Image
axes(AH);
imshow(I);
axis off;

P = char(P);
\%Acquire Data for the Problem
if strcmp(Problem,'Problem1'),
s=1; \%Problem specific variables and graphics scaling
switch Modify
\%As Given
clear Problem1;
D = Problem1;
\%Modify
\%Ask User to Modify the Function Problem1
edit Problem1.m;
\%Provide One Force by specifying End points of Vectors
waitfor(h);
clear Problem1;
D = Problem1;
end

\%Suitable commands for Problem 2
\%Assemble and execute the completed command
elseif strcmp(Problem,'Problem2'),
hndlList=get(gcf,'UserData');
P = get(hndlList(1),'String');
newStrList=get(hndlList(2),'String');
newStrVal=get(hndlList(2),'Value');
Problem=deblank(newStrList(newStrVal,:));
newStrList=get(hndlList(3),'String');
newStrVal=get(hndlList(3),'Value');
D=get(hndlList(3),'Userdata');
AH = hndlList(4);

switch Problem
\%Problem1
\%Problem 1
\%Define Theta and symbols
syms theta FBD;
\%Deltheta = 0.1;
\%Deltatheta
\%Deltatheta = 0.1;
\%Limiting magnitude of Force in member DB in N \=\ mat2str(D.FBDL));
Deltatheta = 0.1;
\%Define DL
DL = (((D.a-D.b*cos(theta))*(D.b*sin(theta)))+ ...
(D.b^2*sin(theta)*cos(theta)))/((sqrt((D.a-D.b*cos(theta))^2+(D.b*sin(theta))^2)));
\%Define f
f = -D.P*D.L*cos(theta)+FBD*DL;
\%Solve
Sol = solve(f,FBD);
function D=Problem1()

% Definition of Data for Virtual Work Problem1

a = 500; % Distance DC in mm
b = 150; % Distance BC in mm
L = 500; % Distance AC in mm
P = 110; % The force P in N applied at A
theta = [30*(pi/180) 150*(pi/180)]; % Range of Theta in rads for which the force in BD has to be found
FBDL = 450; % Limiting force in member BD (in N)

% Convert to structure array
D=struct('a',a,'b',b,'L',L,'P',P,'theta',theta,'FBDL',FBDL);
return;

A.7  AreaMoments.m Matlab GUI file

function AreaMoments(action)
% AreaMoments is a GUI in Matlab that computes the area, centroidal
% coordinates, and inertial moments of an arbitrary user defined polygon.
% This window allows you input a 2D Arbitrary Area (a data text file
% describing the corner coordinates of the Area).
%
% By playing with the popup menus on the right side of the window, the User can
% adjust the type of problem, the solutions required and so on.
%
% The MiniCommand Window in the lower right shows the problem descriptions
% and also solution values.
%
% User m functions called: polyxy.m and Problem1.m
% The user may, and should modify the M-files as they see fit.
% These were developed using MATLAB 7.3.0.267 (R2006b). Although not
% fully tested on older versions, they seem to be backwards
% compatible.
%
% DISCLAIMER:
% "These M-Files are User Contributed Routines which are upon request
% distributed on an "as is" basis. A User Contributed Routine is not a
% product of The Math Works, Inc. and The Math Works or the author assumes
% no responsibility for any errors that may exist in these routines."
% %
% Author: G Saravana Kumar, Year Created: 2008
% All rights reserved with the Author.
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin<1,
    action='initialize';
end;

if strcmp(action,'initialize'),
    oldFigNumber=watchon;
    scrsz = get(0,'ScreenSize');
    figNumber=figure( ...
        'Visible','off', ...
        'NumberTitle','off', ...
        'Color','black', ...
        'Name','AreaMoments', ...
        'Position',scrsz);
    colordef(figNumber,'none');
    AH = axes( ...
        'Units','normalized', ...
        'Position',[0.07 0.45 0.60 0.50];

%===================================
% Set up the Problem statement Window

    top=0.35;
    left=0.05;
    right=0.70;
    bottom=0.05;
    labelHt=0.05;
    spacing=0.005;

    % First, the Problem statement frame
    frmBorder=0.02;
    frmPos=[left-frmBorder bottom-frmBorder ...
        (right-left)+2*frmBorder (top-bottom)+2*frmBorder];
    uicontrol( ...
        'Style','frame', ...
        'Units','normalized', ...
        'Position',frmPos, ...
        'BackgroundColor',[0.50 0.50 0.50];

    % Then the text label
    labelPos=[left top-labelHt (right-left) labelHt];
    uicontrol( ...
        'Style','text', ...
        'Units','normalized', ...
        'Position',labelPos, ...
        'BackgroundColor',[0.50 0.50 0.50];

    % Then the editable text field
    mcwPos=[left bottom (right-left) top-bottom-labelHt-spacing];
    mcwHndl=uicontrol( ...
        'Style','edit', ...
        'HorizontalAlignment','left', ...
        'Units','normalized', ...
        'Max',10, ...
        'BackgroundColor',[1 1 1], ...
        'String','Problem Statement');

% Then the editable text field
    mcwPos=[left bottom (right-left) top-bottom-labelHt-spacing];
    mcwHndl=uicontrol( ...
        'Style','edit', ...
        'HorizontalAlignment','left', ...
        'Units','normalized', ...
        'Max',10, ...
        'BackgroundColor',[1 1 1], ...
        'String','Problem Statement');

% Save this handle for future use
    set(gcf,'UserData',mcwHndl);

%====================================
% Information for all buttons

    labelColor=[0.8 0.8 0.8];
    top=0.95;
    bottom=0.05;
left=0.75;
yInitLabelPos=0.90;
left=0.75;
labelWid=0.20;
labelHt=0.05;
btnWid=0.20;
btnHt=0.05;
% Spacing between the label and the button for the same command
btnOffset=0.003;
% Spacing between the button and the next command's label
spacing=0.05;

% The CONSOLE frame
frmBorder=0.02;
yPos=0.05-frmBorder;
frmPos=[left-frmBorder yPos btnWid+2*frmBorder 0.9+2*frmBorder];
h=uicontrol( ...
    'Style','frame', ...    'Units','normalized', ...
    'Position',frmPos, ...
    'BackgroundColor',[0.50 0.50 0.50]);

% The PROBLEM INPUT command popup button
btnNumber=1;
yLabelPos=top-(btnNumber-1)*(btnHt+labelHt+spacing);
labelStr=' Problem Type';
labelList='Problem1';% 'Problem1|Problem2|...'  
callbackStr='AreaMoments eval';

% Generic label information
labelPos=[left yLabelPos-labelHt labelWid labelHt];
uicontrol( ...        'Style','text', ...        'Units','normalized', ...
        'Position',labelPos, ...
        'BackgroundColor',labelColor, ...
        'HorizontalAlignment','left', ...
        'String',labelStr);

% Generic popup button information
btnPos=[left yLabelPos-labelHt-btnHt-btnOffset btnWid btnHt] ;

% The FORCES command popup button
btnNumber=2;
yLabelPos=top-(btnNumber-1)*(btnHt+labelHt+spacing);
labelStr=' Change Geometry';
labelList='As Given|Modify';
callbackStr='AreaMoments eval';

% Generic label information
labelPos=[left yLabelPos-labelHt labelWid labelHt];
uicontrol( ...        'Style','text', ...        'Units','normalized', ...
        'Position',labelPos, ...
        'BackgroundColor',labelColor, ...
        'HorizontalAlignment','left', ...
        'String',labelStr);

% Generic popup button information
btnPos=[left yLabelPos-labelHt-btnHt-btnOffset btnWid btnHt];
hndl2=uicontrol( ...  
'Style','popup', ...  
'Units','normalized', ...  
'Position',btnPos, ...  
'String',labelList, ...  
'Callback','AreaMoments("solve")');

% The solve button.
uicontrol( ...  
'Style','pushbutton', ...  
'Units','normalized', ...  
'Position',[left bottom+3*btnHt+3*spacing btnWid 2*btnHt], ...  
'String','Solve', ...  
'Callback','AreaMoments("solve")');

% The info button.
uicontrol( ...  
'Style','pushbutton', ...  
'Units','normalized', ...  
'Position',[left bottom+2*btnHt+spacing btnWid 2*btnHt], ...  
'String','Info', ...  
'Callback','AreaMoments("info")');

% The close button.
uicontrol( ...  
'Style','pushbutton', ...  
'Units','normalized', ...  
'Position',left bottom btnWid 2*btnHt], ...  
'String','Close', ...  
'Callback','close(gcf)');

% Uncover the figure
hndlList=[mcwHndl hndl1 hndl2 AH];
watchoff(oldFigNumber);
set(figNumber,'Visible','on', ...  
'UserData',hndlList);
AreaMoments('eval');

elseif strcmp(action,'info'),
  helpwin(mfilename)
elseif strcmp(action,'eval'),
  % Acquire Problem statement and image
  hndl1List=get(gcf,'UserData');
  mcwHndl=hndl1List(1);
  newStrList=get(hndl1List(2),'String');
  newStrVal=get(hndl1List(3),'Value');
  Problem=deblank(newStrList(newStrVal,:));
  newStrList=get(hndl1List(3),'String');
  newStrVal=get(hndl1List(3),'Value');
  Modify=deblank(newStrList(newStrVal,:));
  AH = hndl1List(4);
  %load Problem
  P = textread(strcat(Problem, '.txt'), '%s','whitespace', '');
  %Display Problem Text
  set(mcwHndl,'String',P);
  %Acquire Data for the geometry (As Given / Modified)
  P = char(P);
  if strcmp(Problem,'Problem1'),
    s=1; %Problem specific variables and graphics scaling
    switch Modify
      case 'As Given'
        clear Problem1;
        D = Problem1;
      case 'Modify'
        %Ask User to Modify the Function Problem1
        edit Problem1.m;
  else
    % The solve button.
    uicontrol( ...  
      'Style','pushbutton', ...  
      'Units','normalized', ...  
      'Position',[left bottom+3*btnHt+3*spacing btnWid 2*btnHt], ...  
      'String','Solve', ...  
      'Callback','AreaMoments("solve")');
  end
end


Problem1.m Matlab Input file

function D=Problem1()
  %
  % Definition of Polygonal Data for Area Moments Problem
  % First Boundary (External, Traversed in Counterclockwise, The last Vertex is not repeated

h = helpdlg('Edit the M file containing Data, save and then press OK','Modify Data');
% Provide One Force by specifying End points of Vectors
waitfor(h);
clear Problem1; D = Problem1;
end
%
% Display The Area and Co-ordinate Axis
%Display the Area and Co-ordinate Axis
axes(AH);
for i=1:size(D,1)
  plot([D(i).x,D(i).x(1)],[D(i).y,D(i).y(1)],'-y');
  hold on;
end
hold off;
axis equal; axis tight;
xlabel ('X-axis'); ylabel ('Y-axis');
elseif strcmp(Problem,'Problem2'),
  % Suitable commands for Problem 2
end
set(mcwHndl,'String',P);
set(hndlList(3), 'Userdata', D);
elseif strcmp(action,'solve'),
  % Assemble and execute the completed command
hndlList=get(gcf,'UserData');
P = get(hndlList(1), 'String');
newStrList=get(hndlList(2), 'String');
newStrVal=get(hndlList(2), 'Value');
Problem=deblank(newStrList(newStrVal,:));
D=get(hndlList(3), 'Userdata');
AH = hndlList(4);
AreaM = zeros(1,6);
switch Problem
  case 'Problem1'
    for i=1:size(D,1),
      [area,xbar,ybar,axx,axy,ayy]=polyxy(D(i).x,D(i).y);
      if D(i).Flag ==0,
        AreaM = [AreaM(1)+area,(AreaM(2)*AreaM(1)+xbar*area)/(AreaM(1)+area),
          (AreaM(3)*AreaM(1)+ybar*area)/(AreaM(1)+area),
          AreaM(4)+axx,AreaM(5)+axy,AreaM(6)+ayy];
      else
        AreaM = [AreaM(1)-area,(AreaM(2)*AreaM(1)-xbar*area)/(AreaM(1)-area),
          (AreaM(3)*AreaM(1)-ybar*area)/(AreaM(1)-area),
          AreaM(4)-axx,AreaM(5)-axy,AreaM(6)-ayy];
      end
    end
  end
  case 'Problem2'
    % Suitable commands for Problem 2
end
P = strvcat(P,strcat('Area=',mat2str(AreaM(1))),strcat('Centroid =',mat2str(AreaM(2:3))),strcat('Second Moments XX, XY and YY =',mat2str(AreaM(4:6))));
set(hndlList(1), 'String', P); axis auto;
end;  % if strcmp(action, ...

return;
B1 = [0 0; 10 0; 10 10; 0 10];
% Next Boundary (External)
B2 = [0 10; 10 10; 5 15];
% Next Boundary (Internal)
B3 = [ 1 1; 4 1; 4 4; 1 4];
% Convert to structure array
D=[struct('Flag',0, 'x',B1(:,1), 'y', B1(:,2)); ...
   struct('Flag',0, 'x',B2(:,1), 'y', B2(:,2)); ...
   struct('Flag',1, 'x',B3(:,1), 'y', B3(:,2)); ...]
% Flag = 0 for external and Flag = 1 for internal boundary
return;

A.8 MassMoments.m Matlab GUI file

function MassMoments(action)
% MassMoments is a GUI to Demonstrate the Computation of Volume, Mass, Centroidal
% coordinates, and Inertia Tensor of an arbitrary 3D polyhedron.
% This window allows the user to input a 3D Arbitrary Polyhedron (a data text file
% describing the corner coordinates, faces of the Polyhedron).
% By playing with the popup menus on the right side of the window, the User can
% adjust the type of problem, the solutions required and so on.
% The MiniCommand Window in the lower right shows the problem descriptions
% and also solution values.
% User m functions called: polhdplt.m, polyhedrn.m and Probem1.m
% The user may, and should modify the M-files as they see fit.
% These were developed using MATLAB 7.3.0.267 (R2006b). Although not
% fully tested on older versions, they seem to be backwards
% compatible.
% DISCLAIMER:
% "These M-Files are User Contributed Routines which are upon request
% distributed on an "as is" basis. A User Contributed Routine is not a
% product of The Math Works, Inc. and The Math Works or the author assumes
% no responsibility for any errors that may exist in these routines."
% Author : G Saravana Kumar, Year Created: 2008
% All rights reserved with the Author.

if nargin<1,
    action='initialize';
end;

if strcmp(action,'initialize'),
    oldFigNumber=watchon;
    scrsz = get(0,'ScreenSize');
    figNumber=figure( ...
        'Visible','off', ...
        'NumberTitle','off', ...
        'Color','black', ...
        'Name','MassMoments', ...
        'Position',scrsz);
    colordef(figNumber,'none');
    AH = axes( ...
        'Units','normalized', ...
        'Position',[0.07 0.45 0.60 0.50]);
    % Set up the Problem statement Window
    top=0.35;
    left=0.05;
% Generic label information
labelPos=[left yLabelPos labelHt labelWid labelHt];
uicontrol( ... 'Style','text', ... 'Units','normalized', ... 'Position',labelPos, ... 'BackgroundColor',labelColor, ... 'HorizontalAlignment','left', ... 'String',labelStr);

% Generic popup button information
btnPos=[left yLabelPos labelHt btnOffset btnWid btnHt];
hndl1=uicontrol( ... 'Style','popup', ... 'Units','normalized', ... 'Position',btnPos, ... 'String',labelList, ... 'Callback',callbackStr);

%====================================
% The FORCES command popup button
%====================================
btnNumber=2;
yLabelPos=top-(btnNumber-1)*(btnHt+labelHt+spacing);
labelStr='Change Geometry';
labelList='As Given|Modify';
callbackStr='MassMoments eval';

% Generic label information
labelPos=[left yLabelPos labelHt labelWid labelHt];
uicontrol( ... 'Style','text', ... 'Units','normalized', ... 'Position',labelPos, ... 'BackgroundColor',labelColor, ... 'HorizontalAlignment','left', ... 'String',labelStr);

% Generic popup button information
btnPos=[left yLabelPos labelHt btnOffset btnWid btnHt];
hndl2=uicontrol( ... 'Style','popup', ... 'Units','normalized', ... 'Position',btnPos, ... 'String',labelList, ... 'Callback',callbackStr);

%====================================
% The solve button.
%====================================
untroll( ... 'Style','pushbutton', ... 'Units','normalized', ... 'Position',[left bottom+3*btnHt+3*spacing btnWid 2*btnHt], ... 'String','Solve', ... 'Callback','MassMoments(''solve'')');

% The info button.
untroll( ... 'Style','pushbutton', ... 'Units','normalized', ... 'Position',[left bottom+2*btnHt+spacing btnWid 2*btnHt], ... 'String','Info', ... 'Callback','MassMoments(''info'')');

% The close button.
untroll( ... 'Style','pushbutton', ... 'Units','normalized', ... 'Position',[left bottom btnWid 2*btnHt], ... 'String','Close', ... 'Callback','close(gcf)');
% Uncover the figure
hndlList={[mcwHndl hndl1 hndl2 AH]; watchoff(oldFigNumber);
set(figNumber,'Visible','on', ...
 'UserData',hndlList);
MassMoments('eval');

elseif strcmp(action,'info'),
    helpwin(mfilename)
else if strcmp(action,'eval'),
    % Acquire Problem statement and image
    hndlList=get(gcf,'UserData');
    mcwHndl=hndlList(1);
    newStrList=get(hndlList(2),'String');
    newStrVal=get(hndlList(2),'Value');
    Problem=deblank(newStrList(newStrVal,:));
    newStrList=get(hndlList(3),'String');
    newStrVal=get(hndlList(3),'Value');
    Modify=deblank(newStrList(newStrVal,:));
    AH = hndlList(4);
    %load Problem
    P = textread(strcat(Problem, '.txt'), '%s','whitespace', '');
    %Display Problem Text
    set(mcwHndl,'String',P);
    %Acquire Data for the geometry (As Given / Modified)
    P = char(P);
    if strcmp(Problem,'Problem1'),
        s=1; %Problem specific variables and graphics scaling
        switch Modify
        case 'As Given'
            clear Problem1;D = Problem1;
        case 'Modify'
            %Ask User to Modify the Function Problem1
            edit Problem1.m;
            h = helpdlg('Edit the M file containing Data, save and then press OK','Modify Data');% Provide One Force by specifying End points of Vectors
            waitfor(h);
            clear Problem1;D = Problem1;
        end
    end
    %Display The Polyhedron and Co-ordinate Axis
    axes(AH);
    polhdplt(D.x,D.y,D.z,D.idface,'y');
    axis equal;
else if strcmp(Problem,'Problem2'),
    %Suitable commands for Problem 2
end
set(mcwHndl,'String',P);
set(hndlList(3), 'Userdata', D);
else if strcmp(action,'solve'),
    % Assemble and execute the completed command
    hndlList=get(gcf,'UserData');
    P = get(hndlList(1),'String');
    newStrList=get(hndlList(2),'String');
    newStrVal=get(hndlList(2),'Value');
    Problem=deblank(newStrList(newStrVal,:));
    D=get(hndlList(3),'Userdata');
    AH = hndlList(4);
    switch Problem
    case 'Problem1'
        [v,rc,vrr,irr]=polhedrn(D.x,D.y,D.z,D.idface);
    case 'Problem2'
        %Suitable commands for Problem 2
    end
Problem1.m Matlab Input file

function D=Problem1(),
% Definition of Polyhedral Volume Data

% x,y,z - vectors containing the corner
% indices of the polyhedron
% idface - a matrix in which row j defines the
% corner indices of the j'th face.
% Each face is traversed in a
% counterclockwise sense relative to
% the outward normal. The column
% dimension equals the largest number
% of indices needed to define a face.
% Rows requiring fewer than the
% maximum number of corner indices are
% padded with zeros on the right.

x=[2 2 2 2 2 2 0 0 0 0 0 0]-1;
y=[0 4 4 2 3 3 0 4 4 2 3 3];
z=[0 0 4 1 1 2 0 0 4 1 1 2];

idface=[1 2 3 6 5 4 6 3; 1 3 9 7 0 0 0 0; 1 7 8 2 0 0 0 0; 2 8 9 3 0 0 0 0; 7 9 12 10 11 12 9 8; 4 10 12 6 0 0 0 0; 4 5 11 10 0 0 0 0; 5 6 12 11 0 0 0 0];

% Convert to structure array
D=struct('x',x, 'y',y, 'z',z,'idface',idface); return;
B.1 breakup.m Matlab Library Function file

function [xmag,ymag,xcor,ycor]=breakup(vectors)
% BREAKUP Breaks a standard form force vector into its component parts.
% [XMAG,YMAG,XCOR,YCOR]=BREAKUP(X) Subroutine that breaks a
% multivector load matrix, X, into four column vectors representing the
% x magnitudes, y magnitudes, x coordinates, and y coordinates.
%
% This function is designed as a routine to be called from other
% functions.
% User m functions called: none
%----------------------------------------------
xmag=vectors(:,1);
ymag=vectors(:,2);
xcor=vectors(:,3);
ycor=vectors(:,4);
return;

B.2 comp.m Matlab Library Function file

function [result]=comp(part,req)
% COMP Composite shape routine.
% COMP(PARTS,REQUEST)
% Shape described: A composite shape.
% Datum location: Arbitrary, all shapes are individually placed.
% Input arguments:
% PARTS: An N by 9 matrix containing data on the components of the
% composite shape.
% PARTS=[area, circ, centX, centY, Ix, Iy, X, Y, SIGN]
% The first six pieces of information are easily supplied with other
% shape functions using the "comp" request.
% X and Y are the the distance from the composite datum to the datum
% of each individual shape datum.
% SIGN is either (1) or (-1) with (1) meaning the shape represents
% material and (-1) meaning the shape represents a hole in the
% material.
% Requesting circumference is a feature simply for compliance with other
% shape functions.  The answer arrived at for the circumference is never
% correct due to shared inner borders. Some corrections can be done to
% the answer to find the correct value.
% Requests: (Must be in single quotes)
% 'area': Area of the shape.
% 'circ': Circumference of shape.
% 'Ix': Area moment of inertia about the neutral x axis.
% 'Iy': Area moment of inertia about the neutral y axis.
% 'centX': Distance from datum to centroid in the x direction.
% 'centY': Distance from datum to centroid in the y direction.
% 'comp': All of the above in a 1x6 matrix.
% 'draw': Show the shape graphically.
% User m functions called: none
%----------------------------------------------

req=lower(req);
if     strcmp(req,'area'),  result=sum (part(:,1).*part(:,9));
elseif strcmp(req,'circ'),  result=sum (part(:,2));
elseif strcmp(req,'ix')
    cent=comp(part,'centy');
    d=cent-(part(:,4)+part(:,8));
    result=sum((part(:,5)+part(:,1).*(d.^2)).*part(:,9));
elseif strcmp(req,'iy')
    cent=comp(part,'centx');
end
d=cent-(part(:,3)+part(:,7));
result=sum((part(:,6)+part(:,1).*(d.^2)).*part(:,9));

elseif strcmp(req,'centx')
num=sum((part(:,3)+part(:,7)).*part(:,1).*part(:,9));
den=sum(part(:,1).*part(:,9));
result= num/den;
elseif strcmp(req,'centy')
num=sum((part(:,4)+part(:,8)).*part(:,1).*part(:,9));
den=sum(part(:,1).*part(:,9));
result= num/den;
elseif strcmp(req,'comp')
result(1,1)=comp(part,'area');
result(1,2)=comp(part,'circ');
result(1,3)=comp(part,'centx');
result(1,4)=comp(part,'centy');
result(1,5)=comp(part,'ix');
result(1,6)=comp(part,'iy');
else
    disp ('That is not a valid request, try area, circ, Ix, Iy, centx, centy')
    disp ('You must use single quotes')
end
return;

B.3  crosmat.m Matlab Library Function file

function c=crosmat(a,b)
    c=[a(:,2).*b(:,3)-a(:,3).*b(:,2),...
       a(:,3).*b(:,1)-a(:,1).*b(:,3),...
       a(:,1).*b(:,2)-a(:,2).*b(:,1)];
return;

B.4  cubrange.m Matlab Library Function file

function range=cubrange(xyz,ovrsiz)
    range=cubrange(xyz,ovrsiz)
    range=cubrange(xyz,ovrsiz)
    range=cubrange(xyz,ovrsiz)
    % This function computes the vector cross
    % product for vectors stored in the rows
    % of matrices a and b, and returns the
    % results in the rows of c.
    %
    c=[a(:,2).*b(:,3)-a(:,3).*b(:,2),...
        a(:,3).*b(:,1)-a(:,1).*b(:,3),...
        a(:,1).*b(:,2)-a(:,2).*b(:,1)];

return;
only two columns or the form
[xmin,xmax,ymin,ymax,zmin,zmax]
when xyz has three columns.

if nargin==1, ovrsiz=1; end
pmin=min(xyz); pmax=max(xyz); pm=(pmin+pmax)/2;
pd=max(ovrsiz/2*(pmax-pmin));
if length(pmin)==2
range=pm([1,1,2,2])+pd*[-1,1,-1,1];
else
range=pm([1 1 2 2 3 3])+pd*[-1,1,-1,1,-1,1];
end
return;

B.5 deg2xy.m Matlab Library Function file

function [vector]=deg2xy(inputs)

% DEG2XY Converts vectors in degree angles to standard form.

% [VECTOR]=DEG2XY (INPUTS) Routine takes any number of force
% vectors described by their angle and magnitude along with an optional
% set of coordinates and returns the force vector in standard form.

% INPUTS: [ANGLE,MAG,XCOR,YCOR]

% User m functions called: DR.m
-----------------------------------------------

[r,c]=size(inputs);
if c==4
ycor=inputs(:,4); % pull coordinates out of input matrix
xcor=inputs(:,3); % pull coordinates out of input matrix
end
mag=inputs(:,2); % pull magnitudes out of input matrix
angle=inputs(:,1); % pull angles out of input matrix
if c==2 % if coordinates were not given
xcor=zeros(r,1); ycor=zeros(r,1); % coordinates default to origin
end
angle=DR(angle); % convert to radians
xmag=cos(angle).*mag; % equation 1.2
ymag=sin(angle).*mag; % equation 1.2
vector=[xmag,ymag,xcor,ycor]; % reassemble complete answer
return;

B.6 diagram.m Matlab Library Function file

function [y]=diagram (x,option,mag,place)

% DIAGRAM Creates vectors for use in plotting of diagrams.
% DIAGRAM(X,OPTION,MAG,PLACE) will create a data vector the same length as
% the input X. The data will be created to one of three options:
% Options:
% 'point' all data points that correspond to X>=PLACE are set equal to
% MAG. This is useful in shear diagrams when a point load is used.
% 'linear' all data points that correspond to PLACE(1)<=X<=PLACE(2) are
% set equal to the linear interpolation of the values MAG(1) and
% MAG(2). This is useful when describing the area of a shaft that is
% constant over a set length or that changes linearly over a set
% length. This is not for distributed loads.
% 'distributed' all data points are created to reflect the load from a
option=lower(option);
y=zeros(size(x));

if strcmp(option,'point')
    VI=find(x>=place); %Valid Indices
    y(VI)=mag;
elseif strcmp(option,'linear')
    VI=find(x>=place(1) & x<=place(2)); %Valid Indices
    y(VI)=(x(VI)-place(1))/(place(2)-place(1))*(mag(2)-mag(1))+mag(1);
else
    disp ('Use a proper loading option: "point", "linear", or "distributed")
    return;
end

B.7 diagramintegral.m Matlab Library Function file

function [area]=diagramintegral(x,y)
% DIAGRAMINTEGRAL Integral of the given numerical data.
% DIAGRAMEGINTEGRAL(X,Y) finds the integral of the data vector Y that
% corresponds to the indices X.  The trapezoid method is used for
% integration. The routine is intended to work with the output from
% DIAGRAM to create a moment diagram from a shear diagram, or other such
% integrations.
% User m functions called: none
%---------------------------------------------------------------

if length(x)~=length(y)
    disp('Vectors must be the same length."
    return
end

DeltaX=x(2)-x(1);
n=length(x);

subarea(1)=0;
for gapli=2:n %Generic All Purpose Looping Index
    subarea(gapli)=(y(gapli-1)+y(gapli))*DeltaX/2;
end

area=cumsum(subarea);
return;

B.8 distload.m Matlab Library Function file

function [force, placement]=distload(ho,ht,x)
% DISTLOAD Converts a linearly distributed load to a point force.
% [FORCE, PLACEMENT]=DISTLOAD(H1,H2,X) Given the two magnitudes H1 and H2
% that a linearly distributed load varies between, and the length of
% application, the routine will find the magnitude and location of the
% equivalent force.
% 
% User m functions called: vertrap.m
%----------------------------------------------

if ho >= ht % left side is bigger
    force = vertrap(ho,x,ht,(ho-ht),'area'); % find area
    placement = vertrap(ho,x,ht,(ho-ht),'centX'); % find centroid
else % right side is bigger
    force = vertrap(ht,(-x),ho,(ht-ho),'area'); % find area
    placement = x + (vertrap(ht,(-x),ho,(ht-ho),'centX')); % find centroid
end
return;

B.9  DR.m Matlab Library Function file

function [radians]=DR(degrees)
%   DR Changes a matrix of degree measure to a matrix of radian measure.
%   DR(X) is the radian equivalent of the elements of X.
%
%   User m functions called: none
%----------------------------------------------

radians=degrees*pi/180;
return;

B.10 expandaxis.m Matlab Library Function file

function []=expandaxis(perleft, perright, perdown, perup)
%   EXPANDAXIS Extends the current axis in any or all directions.
%   EXPANDAXIS(left, right, down, up) Each of the direction will be expanded
%   by the specified percent. For example if Left were specified as 30 then
%   the upper bound of the axis would be increased by 30% of the vertical
%   distance across the horizontal axis.
%   % If no argument is given, the routine will default to 10% in the
%   % unspecified directions.
%   % User m functions called: none
%----------------------------------------------

if nargin<4
    perup=10;
end
if nargin<3
    perdown=10;
end
if nargin<2
    perright=10;
end
if nargin<1
    perleft=10;
end

edges=axis;
left=edges(1);
right=edges(2);
bottom=edges(3);
top=edges(4);

hordist=right-left;
verdist=top-bottom;
newleft = left - (hordist * perleft / 100);
newright = right + (hordist * perright / 100);
newbottom = bottom - (verdist * perdown / 100);
newtop = top + (verdist * perup / 100);

axis([newleft newright newbottom newtop]);
return;

B.11 hortria.m Matlab Library Function file

function [result] = hortria(b,h,p,req)
%HORTRIA Horizontal triangle shape routine.
% HORTRIA(BASE,HEIGHT,P,REQUEST)
%
% Shape described: Triangle with a horizontal base.
% Datum location: Leftmost point of base.
% Input arguments:
% BASE: Length of base (base must be a horizontal line).
% HEIGHT: Vertical distance between base and vertex may be negative.
% P: Horizontal distance from the datum to lower edge of small edge. May
% be negative.
% Requests: (Must be in single quotes)
% 'area': Area of the shape.
% 'circ': Circumference of shape.
% 'Ix': Area moment of inertia about the neutral x axis.
% 'Iy': Area moment of inertia about the neutral y axis.
% 'centX': Distance from datum to centroid in the x direction.
% 'centY': Distance from datum to centroid in the y direction.
% 'comp': All of the above in a 1x6 matrix.
% 'draw': Show the shape graphically.
%
% User m functions called: vertria.m, titleblock.m makecol.m and expandaxis.m

req = lower(req);

origH = h;
h = abs(h);
if b < 0
    disp('Base must be a positive length')
    return
end

if strcmp(req,'area')
    result = b*h/2;
elseif strcmp(req,'circ')
    result = sqrt(h^2 + p^2) + sqrt(h^2 + (b-p)^2) + b;
elseif strcmp(req,'ix')
    if p == 0 | p == b % right triangle
        result = b*h^3/36;
    elseif p < 0 % vertex extend past left edge of base.
        midh = b*h/(abs(p)+b);
        lsi = vertria(midh,p,h,'iy');
        lsa = vertria(midh,p,h,'area');
        lsd = vertria(midh,p,h,'centX');
        rsa = vertria(midh,b,0,'iy');
        rsd = vertria(midh,b,0,'area');
        centroid = hortria(b,h,p,'centX');
        result = lsi + lsa*(centroid-lsd)^2 + rsa*(centroid-rsd)^2;
    end
elseif p > b % vertex extend past right edge of base.
    midh = b*h/p;
    lsi = vertria(midh,-b,0,'iy');
    lsa = vertria(midh,-b,0,'area');


lsd=vertria(midh,-b,0,'centx')+b;
sri=vertria(midh,p-b,h,'iy');
rsa=vertria(midh,p-b,h,'area');
rsd=vertria(midh,p-b,h,'centx')+b;
centroid=hortria(b,h,p,'centx');
result=lsi+lsa*(centroid-lsd)^2+rsi+rsa*(centroid-rsd)^2;
end
if p>0 & p<b % vertex is over the base
lsi=vertria(h,-p,0,'iy');
lsa=vertria(h,-p,0,'area');
lsd=vertria(h,-p,0,'centx')+p;
rsi=vertria(h,p-0,'iy');
rsa=vertria(h,b-p,0,'area');
rsd=vertria(h,b-p,0,'centx')+p;
centroid=hortria(h,p,'centx');
result=lsi+lsa*(centroid-lsd)^2+rsi+rsa*(centroid-rsd)^2;
end
elseif strcmp(req,'centx')
if p<0  % vertex extend past left edge of base.
midh=b*h/(abs(p)+b);
lsa=vertria(midh,p,h,'area');
lsd=vertria(midh,p,h,'centx');
rsa=vertria(midh,b,0,'area');
rsd=vertria(midh,b,0,'centx');
wa=hortria(b,h,p,'area');
result=((lsa*lsd)+(rsa*rsd))/wa;
end
if p==0 % right triangle
result=b/3;
end
if p==b % right triangle
result=b*2/3;
end
if p>b % vertex extend past right edge of base.
midh=b*h/p;
lsa=vertria(midh,-b,0,'area');
lsd=vertria(midh,-b,0,'centx')+b;
rsa=vertria(midh,b,0,'area');
rsd=vertria(midh,b,0,'centx')+b;
wa=hortria(h,p,'area');
result=((lsa*lsd)+(rsa*rsd))/wa;
end
if p>0 & p<b % vertex is over the base
lsa=vertria(h,-p,0,'area');
lsd=vertria(h,-p,0,'centx')+p;
rsa=vertria(h,b-p,0,'area');
rsd=vertria(h,b-p,0,'centx')+p;
wa=hortria(h,p,'area');
result=((lsa*lsd)+(rsa*rsd))/wa;
end
else if strcmp(req,'centy')
result=origH/3;
elseif strcmp(req,'comp')
result(1,1)=hortria(b,origH,p,'area');
result(1,2)=hortria(b,origH,p,'circ');
result(1,3)=hortria(b,origH,p,'centx');
result(1,4)=hortria(b,origH,p,'centy');
result(1,5)=hortria(b,origH,p,'ix');
result(1,6)=hortria(b,origH,p,'iy');
elseif strcmp(req,'draw')
xcentroid=hortria(b,origH,p,'centx');
ycentroid=hortria(b,origH,p,'centy');
X=[0 0 p 0];
Y=[0 0 origH 0];
fill(X,Y,'r')
hold on;
CoA=strvcat('Area','Circumference','Centroid X','Centroid Y','Ix','Iy');
ColB=makecol(hortria(b,origH,p,'comp'));
plot (xcentroid,ycentroid,'ko',0,0,'g*')
B.12 interpolate.m Matlab Library Function file

function [outvalue]=interpolate(x,y,invalue)
% INTERPOLATE Linear interpolation for a given value.
% INTERPOLATE(X,Y,XVALUE) With a vector of X and Y values that correspond to
% one another, the linear interpolation of the YVALUE that corresponds to the
% given XVALUE will be found.
% User m functions called: none
%----------------------------------------------

if invalue>max(x) | invalue<min(x)
disp('That value is not in the range that can be interpolated.')
end
biggerindex=min(find(x>=invalue));
smallerindex=max(find(x<=invalue));
X2=x(biggerindex);
Y2=y(biggerindex);
X1=x(smallerindex);
Y1=y(smallerindex);
if X2==X1
outvalue=Y1;
else
outvalue=((invalue-X1)/(X2-X1))*(Y2-Y1)+Y1;
end
return;

B.13 makecol.m Matlab Library Function file

function [Column]=makecol(data,formatno);
% Used with titleblock, from the shape routines.
% User m functions called: none
%----------------------------------------------

if nargin<2
formatno=4;
end
[NoRows, NoCols]=size(data);
Column=num2str(data(1),formatno);
for gapli=2:NoRows
Column=strvcat(Column,num2str(data(gapli),formatno));
end
return;

B.14 plotSMD.m Matlab Library Function file

function []=plotSMD(x,shear,moment,AH)
% PLOTSMD Plots a Shear Moment diagram.
PLOTSMD(X,SHEAR,MOMENT,AH) is a quick routine to show the SHEAR, MOMENT diagram on the AH figure axis. This routine can and should be modified to specific needs.

User m functions called: none

if nargin ==4
  subplot(AH)
end
subplot(3,2,1)
plot ([0,x],[0,shear])
title ('Shear')

subplot(3,2,3)
plot ([0,x],[0,moment])
title ('Moment')

B.15 polhdplt.m Matlab Library Function file

function polhdplt(x,y,z,idface,colr)

% polhdplt(x,y,z,idface,colr)

% This function makes a surface plot of an arbitrary polyhedron.

% x,y,z - vectors containing the corner indices of the polyhedron
% idface - a matrix in which row j defines the corner indices of the j'th face.
% Each face is traversed in a counterclockwise sense relative to the outward normal. The column dimension equals the largest number of indices needed to define a face.
% Rows requiring fewer than the maximum number of corner indices are padded with zeros on the right.
% colr - character string or a vector defining the surface color

% User m functions called: cubrange.m

if nargin<5, colr=[1 0 1]; end
nf=size(idface,1);
v=cubrange([x(:),y(:),z(:)],1.1);
for k=1:nf
  i=idface(k,:); i=find(i>0);
  xi=x(i); yi=y(i); zi=z(i);
  fill3(xi,yi,zi,colr); hold on;
end
axis(v); grid on;
xlabel('x axis'); ylabel('y axis'); zlabel('z axis');
hold off;
return;

B.16 polhedm.m Matlab Library Function file

function [v,rc,vrr,irr]=polhedrn(x,y,z,idface)
% [v,rc,vrr,irr]=polhedrn(x,y,z,idface)
%
% This function determines the volume,
% centroidal coordinates and inertial moments
% for an arbitrary polyhedron.
%
% x,y,z  - vectors containing the corner
% indices of the polyhedron
% idface - a matrix in which row j defines the
% corner indices of the j'th face.
% Each face is traversed in a
% counterclockwise sense relative to
% the outward normal. The column
% dimension equals the largest number
% of indices needed to define a face.
% Rows requiring fewer than the
% maximum number of corner indices are
% padded with zeros on the right.
%
% v      - the volume of the polyhedron
% rc     - the centroidal radius
% vrr    - the integral of R*R'*d(vol)
% irr    - the inertia tensor for a rigid body
% of unit mass obtained from vrr as
% eye(3,3)*sum(diag(vrr))-vrr
%
% User m functions called: pyramid.m
%----------------------------------------------

r=[x(:),y(:),z(:)]; nf=size(idface,1);
v=0; vr=0; vrr=0;
for k=1:nf
i=idface(k,:); i=i(find(i>0));
[u,ur,urr]=pyramid(r(i,:));
v=v+u; vr=vr+ur; vrr=vrr+urr;
end
rc=vr/v; irr=eye(3,3)*sum(diag(vrr))-vrr;
return;

B.17 polyxy.m Matlab Library Function file

function [area,xbar,ybar,axx,axy,ayy]=polyxy(x,y)
%
% [area,xbar,ybar,axx,axy,ayy]=polyxy(x,y)
%
% This function computes the area, centroidal
% coordinates, and inertial moments of an
% arbitrary polygon.
%
% x,y  - vectors containing the corner
% coordinates. The boundary is
% traversed in a counterclockwise
% direction
%
% area - the polygon area
% xbar,ybar - the centroidal coordinates
% axx   - integral of x^2*dx*dy
% axy   - integral of x*y*dx*dy
% ayy   - integral of y^2*dx*dy
%
% User m functions called: none
%----------------------------------------------
n=1:length(x); n1=n+1;
x=[x(:);x(1)]; y=[y(:);y(1)];
a=(x(n).*y(n1)-y(n).*x(n1))';
area=sum(a)/2; a6=6*area;
xbar=a*(x(n)+x(n1))/a6; ybar=a*(y(n)+y(n1))/a6;
axy=a*(x(n).*(2*y(n)+y(n1))+x(n1).* ... 
(2*y(n1)+y(n)))/24;
axx=a*(x(n).^2+x(n).*x(n1)+x(n1).^2)/12;
return;

B.18 pyramid.m Matlab Library Function file

function [v,vr,vrr,h,area,n]=pyramid(r)
%
% [v,vr,vrr,h,area,n]=pyramid(r)
% 
% This function determines geometrical
% properties of a pyramid with the apex at the
% origin and corner coordinates of the base
% stored in the rows of r.
%  
% r - matrix containing the corner
% coordinates of a polygonal base stored
% in the rows of matrix r.
% 
% v - the volume of the pyramid
% vr - the first moment of volume relative to
% the origin
% vrr - the second moment of volume relative
% to the origin
% h - the pyramid height
% area - the base area
% n - the outward directed unit normal to
% the base
% 
% User m functions called: crosmat.m and polyxy.m
%
ns=size(r,1);
na=sum(crosmat(r,r([2:ns,1],:)))'/2;
area=norm(na); n=na/area; p=null(n');
i=p(:,1); j=p(:,2);
if det([p,n])<0, j=-j; end;

cor=r1=r1'+xc*i+yc*j;

B.19 rad2xy.m Matlab Library Function file

function [vector]=rad2xy(inputs)
%\textsc{rad2xy} Converts vectors in radian angles to standard form.
% \textsc{rad2xy} (\textsc{inputs}) Routine takes any number of force vectors described by
% their angle and magnitude along with an optional set of coordinates and
% returns the force vector in standard form.
% 
% \textsc{inputs}: [ANGLE,MAG,XCOR,YCOR]
% User m functions called: none
%
[r,c]=size(inputs);
if c==4  % if coordinates were given
ycor=inputs(:,4); % pull coordinates out of input matrix
% pull coordinates out of input matrix
end

% pull magnitudes out of input matrix
angle=inputs(:,1); % pull angles out of input matrix
if c==2 % if coordinates were not given
    xcor=zeros(r,1); ycor=zeros(r,1); % coordinates default to origin
end

xvec=cos(angle).*mag;
yvec=sin(angle).*mag;

vector=[xvec,yvec,xcor,ycor]; % reassemble complete answer
return;

B.20 RD.m Matlab Library Function file

function [degrees]=RD(radians)
    % RD Changes a matrix of radian measure to a matrix of degree measure.
    % RD(X) is the radian equivalent of the elements of X.
    % User m functions called: none
    %----------------------------------------------
    degrees=radians*180/pi;
    return;

B.21 reaction.m Matlab Library Function file

function [force,moment]=reaction(vectors,coords,couples)
    % REACTION Finds the reaction force and moment needed to balance a force.
    % [FORCE,MOMENT]=REACTION(AF,[X Y], COUPLES) Given an Applied Force matrix
    % in standard multi vector format and the coordinates of the unknown
    % reaction force, the routine will return the reaction force and moment
    % at that point. If no couple is specified, it defaults to zero.
    % User m functions called: sumforce.m and summoments.m
    %----------------------------------------------
    if nargin==2 % if couples not given
        couples=0; % couples default to zero
    end;
    [force,unusedcouple]=sumforce(vectors); % find sumation of vectors
    moment=-(sum(couples)+summoment(vectors,coords)); % find sum of moments
    force=-force; % since it is a reaction force must be negated
    force(3)=coords(1); % move vector to proper spot
    force(4)=coords(2); % move vector to proper spot
    return;

B.22 rectangl.m Matlab Library Function file

function [result]=rectangl(b,h,req)
    % RECTANGLE Rectangular shape routine.
    % [result]=rectangl(BASE,HEIGHT,REQUEST)
    % Shape described: A rectangle.
    % Datum location: Bottom left corner.
    % Input arguments:
    % base: Distance between the vertical sides.
    % height: Distance between the horizontal sides.
% Requests: (Must be in single quotes)
% 'area': Area of the shape.
% 'circ': Circumference of shape.
% 'Ix': Area moment of inertia about the neutral x axis.
% 'Iy': Area moment of inertia about the neutral y axis.
% 'centX': Distance from datum to centroid in the x direction.
% 'centY': Distance from datum to centroid in the y direction.
% 'comp': All of the above in a 1x6 matrix.
% 'draw': Show the shape graphically.
% Users m functions called: titleblock.m, makecol.m and expandaxis.m

req=lower(req);
if     strcmp(req,'area')  result=b*h;
elseif strcmp(req,'circ')  result=2*(b+h);
elseif strcmp(req,'ix')    result=b*h^3/12;
elseif strcmp(req,'iy')    result=h*b^3/12;
elseif strcmp(req,'centx') result=b/2;
elseif strcmp(req,'centy') result=h/2;
elseif strcmp(req,'comp')
    result(1,1)=rectangl(b,h,'area');
    result(1,2)=rectangl(b,h,'circ');
    result(1,3)=rectangl(b,h,'centx');
    result(1,4)=rectangl(b,h,'centy');
    result(1,5)=rectangl(b,h,'ix');
    result(1,6)=rectangl(b,h,'iy');
else
    disp ('That is not a valid request, try area, circ, Ix, Iy, centX,)
    disp ('centY, comp.')
    disp ('You must use single quotes')
end

return;

B.23 showvect.m Matlab Library Function file

function []=showvect(vectors, scale, colour, annotation)
% SHOWVECT Draws a simple diagram showing the input vectors.
% SHOWVECT(vectors) Shows all the input vectors on the same coordinate axis.
% Heads of vectors are designated with an "X" while tails are marked with "O".
% Since the input vectors can be any set of vectors in standard format it is
% possible to combine the input and output of solving functions to look at the
% relationship between the two. The simplest way to do this would be
% SHOWVECT ([input; output])
% Use scale to scale the vector with respect to magnitude. (scalar value)
% Use colour to specify the colour of the vectors, SHOWVECT(vectors, scale, %
% colour).
% Use annotations to place annotations for vectors by giving character string,
% SHOWVECT(vectors, scale, colour, annotation).
% Use AXIS (’equal’) to scale the drawing properly, may cause the vectors to
% run off the edge of the plot. If the all of the vectors do not appear, run
% EXPANDAXIS.
%
% User m functions called: breakup.m
%----------------------------------------------

[xmag, ymag, xcor, ycor] = breakup(vectors);
% xmin = min(min([xcor, xcor + xmag])); % leftmost edge
% xmax = max(max([xcor, xcor + xmag])); % rightmost edge
% ymin = min(min([ycor, ycor + ymag])); % lower edge
% ymax = max(max([ycor, ycor + ymag])); % upper edge
% xmar = max([abs(xmax - xmin) * .2, 1]); % margin of 20% of width
% ymar = max([abs(ymax - ymin) * .2, 1]); % margin of 20% of height
% xmin = xmin - xmar; % add a margin around plot
% xmax = xmax + xmar; % add a margin around plot
% ymin = ymin - ymar; % add a margin around plot
% ymax = ymax + ymar; % add a margin around plot
%clf %clear figure
hold on % stops automatic clearing of plot
S = [];
if nargin < 3, colour = 'r';
elseif nargin > 3, S = annotation; end
if isempty(S),
temp = sqrt(xmag.^2 + ymag.^2);
for i = 1:length(temp),
S = strvcat(S, num2str(temp(i)));
end
end
for i = 1:length(xmag) % do once for each vector to be plotted
xhead = xcor(i) + xmag(i) * scale;
xtail = xcor(i);
yhead = ycor(i) + ymag(i) * scale;
ytail = ycor(i);
plot(xtail, ytail, strcat(colour, 'o'), 'MarkerSize', 12);
plot([xtail, xhead, ytail, yhead], strcat(colour, '-'), 'LineWidth', 2);
plot(xhead, yhead, strcat(colour, 'x'), 'MarkerSize', 12);
text(xhead + 10, yhead + 10, S(i,:), 'FontSize', 18, 'Color', 'k');
end
hold off % starts automatic clearing of plot
% axis ([xmin xmax ymin ymax]) % sets scale
% showx; showy
return;

B.24 showx.m Matlab Library Function file

function [] = showx(x, colour)
% SHOWX Draws a line across the current axis.
% SHOWX(X, COLOR) draws a line across the current axis at the specified X in
% the specified COLOR. If no X is specified, the line will be drawn at X=0.
%
% User m functions called: none
%----------------------------------------------

if nargin < 2
    colour = 'k';
end
if nargin < 1
    x = 0;
end

axisvalues = axis;
xmin = axisvalues(1);
xmax = axisvalues(2);
if ishold

return;
for gapli = 1 : length(x)
    plot([xmin xmax],[x(gapli) x(gapli)],colour)
end
else
    hold on
for gapli = 1 : length(x)
    plot([xmin xmax],[x(gapli) x(gapli)],colour)
end
    hold off
end
return;

B.25 showy.m Matlab Library Function file

function []=showy(y,colour)
%SHOWY Draws a line across the current axis.
% SHOWY(X,COLOR) draws a line across the current axis at the specified Y in
% the specified COLOR. If no Y is specified, the line will be drawn at Y=0.
% % User m functions called: none
%----------------------------------------------
if nargin<2
    colour='k';
end
if nargin<1
    y=0;
end
axisvalues=axis;
ymin=axisvalues(3);
ymax=axisvalues(4);
if ishold
    for gapli = 1 : length(y)
        plot([y(gapli) y(gapli)],[ymin ymax],colour)
    end
else
    hold on
    for gapli = 1 : length(y)
        plot([y(gapli) y(gapli)],[ymin ymax],colour)
    end
    hold off
end
return;

B.26 ST.m Matlab Library Function file

function [F,U,R]=ST(D)
%   Compute the forces in each members of a truss
% % User m functions called: none
%----------------------------------------------
w=size(D.Re);S=zeros(3*w(2));U=1-D.Re;f=find(U);
for i=1:size(D.Con,2)
    H=D.Con(:,i);C=D.Coord(:,H(2))-D.Coord(:,H(1));Le=norm(C);
    T=C/Le;s=T*T';G=D.E(i)*D.A(i)/Le;Tj(:,i)=G*T;
    e=[3*H(1)-2:3*H(1),3*H(2)-2:3*H(2)];S(e,e)=S(e,e)+G*[s -s;-s s];
    end
U(f)=S(f,f)
F=sum(Tj.*(U(:,D.Con(2,:))-U(:,D.Con(1,:))));
R=reshape(S*U(:,),w);R(f)=0;
return;

B.27 sumforce.m Matlab Library Function file
function [resultant,couple]=sumforce(vectors)
% SUMFORCE Sums a set of vectors to one force vector and a couple.
% [FORCE, COUPLE]=SUMFORCE(VECTORS) Given a set of known vectors in
% standard multi vector format the routine will return the sum of those
% vectors as a single vector acting through a point so that there is no
% couple needed to balance the original. If such a vector placement is
% not possible, a non-zero value for the couple will be returned. This
% usually occurs due to a force couple being formed.
%
% User m functions called: breakup.m
%--------------------------------------------------------------

[xmag,ymag,xcor,ycor]=breakup(vectors); % call subroutine
couple=0; % set couple to zero
xres=sum(xmag); % x resultant
yres=sum(ymag); % y resultant
if xres==0 % if no x resultant
  ycen=0; % move resultant onto x axis
  couple=couple+sum(xmag.*(-ycor)); % check for a couple
else % there is an x resultant
  ycen=sum(xmag.*ycor)/xres; % move x res to maintain equal moment
end
if yres==0 % if no y resultant
  xcen=0; % move resultant onto y axis
  couple=couple+sum(ymag.*xcor); % check for a couple
else % there is a y resultant
  xcen=sum(ymag.*xcor)/yres; % move y res to maintain equal moment
end
resultant=[xres,yres,xcen,ycen]; % reassemble resultant matrix
return;

B.28 summoment.m Matlab Library Function file

function [moment]=summoment(vectors, coords)
% SUMMOMENT Solves for the moment caused by a set of forces.
% [moment]=SUMMOMENT(VECTORS, [X,Y]) Given a set of known force
% vectors in standard multi vector format, the routine will return the
% resultant moment as seen from the coordinates passed in with the known
% force vectors. If no coordinates are specified then the moment is
% calculated about the origin. Right hand rule for sign convention.
%
% User m functions called: breakup.m
%--------------------------------------------------------------

if nargin==1 % if coordinates are not included
  xpos=0; % coordinates default to zero
  ypos=0; % coordinates default to zero
else
  xpos=coords(1);
  ypos=coords(2);
end
[xmag,ymag,xcor,ycor]=breakup(vectors); % call subroutine
xmoment=sum(xmag.*(ypos-ycor)); % X forces times moment arm
ymoment=sum(ymag.*(xcor-xpos)); % Y forces times moment arm
moment=xmoment+ymoment; % sum both moments
return;

B.29 TB.m Matlab Library Function file

function TB(D,AH)
% Display Beam and Forces in the axis AH
% Scale
% User m functions called: showvect.m
s = 1;
axes(AH);
plot([0 D.Length],[0 0],'o-y');
showvect(D.Force,s);
axis(axis);
drawnow;
return;

**B.30 threevector.m Matlab Library Function file**

function [reactions]=threevector(knowns, unknowns, couples)
% THREEVECTOR Solves for three force vectors of known direction only.
% THREEVECTOR (KNOWNS, UNKNOWNS, COUPLES) Routine takes a rigid body
% acted upon by a set of known load vectors and balanced by a set of three
% forces of known direction and unknown magnitude, and solves for the
% previously unknown magnitudes. The answer is returned in standard
% multivector format. Angles are to be given in radians.
% KNOWNS matrix is in standard format
% UNKNOWNS: [ANGLE1, X1, Y1; ANGLE2, X2, Y2; ANGLE3, X3, Y3];
% COUPLES matrix is optional [COUPLE1, COUPLE2, COUPLE3 ...];
% User m functions called: summoment.m

if nargin==2 couples=0; end % couples defaults to zero

coef(1,:)=cos(unknowns(:,1)');
coef(2,:)=sin(unknowns(:,1)');
coef(3,:)=unknowns(:,2)'.*coef(2,:)-unknowns(:,3)'.*coef(1,:);
answ(1,:)=(-1)*sum(knowns(:,1));
answ(2,:)=(-1)*sum(knowns(:,2));
answ(3,:)=(-1)*(summoment(knowns)+couples);

valu=inv(coef)*answ; % solving a system of three equations three unknowns

reactions(:,1)=coef(1,:)'.*valu;
reactions(:,2)=coef(2,:)'.*valu;
reactions(:,3:4)=unknowns(:,2:3);
return;

**B.31 titleblock.m Matlab Library Function file**

function []=titleblock(columnA,columnB)
% TITLEBLOCK Adds two columns of text within the axis border.
% TITLEBLOCK(COL1,COL2) Adds two columns of text to axis border.
% Automatically expands the axis to an appropriate setting.
% User m functions called: expandaxis.m

edges=axis;
left=edges(1);
right=edges(2);
bottom=edges(3);
top=edges(4);

hordist=right-left;
verdist=top-bottom;

X1=left+(hordist/10);
X2=left+(hordist/2);

[numrowsA numcolsA]=size(columnA);
for gapli=numrowsA:-1:1;
text(X1,(gapli*verdist/20)+top,columnA(numrowsA+1-gapli,:));
end

[numrowsB numcolsB]=size(columnB);
for gapli=numrowsB:-1:1;
text(X2,(gapli*verdist/20)+top,columnB(numrowsA+1-gapli,:));
end

expandaxis(0,0,0,max([numrowsA,numrowsB])*6);
return;

B.32 TP.m Matlab Library Function file

function TP(D)
%   Display Truss
%   Offset for placing Text, depends of the Image Dimesion
%   User m functions called: none
%----------------------------------------------
s = 20;
C=[D.Coord];e=D.Con(1,:);f=D.Con(2,:);
for i=1:3
    M=[C(i,e);C(i,f);repmat(NaN,size(e))];X(:,i)=M(:);
end
plot3(X(:,1),X(:,2),X(:,3),'o-y');axis('equal'); axis off;
for i=1:size(D.Coord,2)
    text(D.Coord(1,i)+s,D.Coord(2,i)+s,D.Coord(3,i),num2str(i));
end
if D.Re(3,:)==1;view(2);end
return;

B.33 twovector.m Matlab Library Function file

function [reactions]=twovector(knowns, unknowns)
%   TWOVECTOR Solves for two force vectors of known direction only.
%   TWOVECTOR(KNOWNS, UNKNOWNS)  Routine takes a single point acted upon by
%   a set of known load vectors and balanced by a set of two forces of known
%   direction and unknown magnitude, and solves for the previously unknown
%   magnitudes. The answer is returned in standard multi vector format.
%   Angles are to be given in radians. This routine is particularly designed
%   for truss problems.
%   %
%   % KNOWNS matrix is in standard format
%   % UNKNOWNS matrix [ANGLE1, ANGLE2];
%   %
%   % User m functions called: breakup.m
%----------------------------------------------
[xmag ymag xcor ycor]=breakup(knowns);
flagx = xcor == mean(xcor)*ones(size(xcor));
flagy = ycor == mean(ycor)*ones(size(ycor));
if (flagx | flagy)
    disp ('In twovector.m all vectors must originate from the same point')
    return
end % if not all from same point
knowns=[sum(xmag) sum(ymag) xcor(1) ycor(1)];
[xmag ymag xcor ycor]=breakup(knowns); 
k=sqrt(xmag^2+ymag^2);
angle=atan2(ymag, xmag);
alpha=unknowns(1);
beta=unknowns(2);
coef=[cos(alpha) cos(beta);sin(alpha) sin(beta)];
answ=-k*[cos(angle);sin(angle)];
mag=inv(coef)*answ;
function [result]=vertrap(b,h,a,p,req)
% VERTRAP Horizontal trapezoid shape routine.
% VERTRAP(BASE,HEIGHT,A,P,REQUEST)
%
% Shape described: Trapezoid with a vertical base.
% Datum location: Lowermost point of base.
%
% Input arguments:
% base: Length of base (base is the larger of the vertical sides).
% height: Distance between the vertical sides, may be negative.
% a: Length of the shorter of the vertical sides.
% p: Vertical distance from the datum to lower edge of small edge.
%
% Requests: (Must be in single quotes)
% 'area': Area of the shape.
% 'circ': Circumference of shape.
% 'Ix': Area moment of inertia about the neutral x axis.
% 'Iy': Area moment of inertia about the neutral y axis.
% 'centX': Distance from datum to centroid in the x direction.
% 'centY': Distance from datum to centroid in the y direction.
% 'comp': All of the above in a 1x6 matrix.
% 'draw': Show the shape graphically.
%
% User m functions called: comp.m, makecol.m, vertria.m,
% expandaxis.m and titleblock.m
%----------------------------------------------

req=lower(req);

if (a > b)
    disp ('"A" value may not be larger than "B" value. Remember to use
')
    disp ('proper sign convention for "H"
')
    return
end

if (p+a > b) | (p < 0)
    disp ('"A" side of trapezoid may not extend beyond the boundary
')
    disp ('of the base
')
    return
end

Origh=h;

h=abs(h);

IsPos= (Origh==h)*2-1;
% breaks up the trapezoid into two triangles and a % rectangle to be used in
% the comp shape routine.

part(1,:)=[vertria(b-p-a,h,0,'comp'),0,(a+p),1];
part(2,:)=[rectangl(h,a,'comp'),0,p,1];
part(3,:)=[vertria(p,h,p,'comp'),0,0,1];

if strcmp(req,'area') result=h*(a+b)/2;
else if strcmp(req,'circ') result=sqrt(p^2+h^2)+a+sqrt(h^2+(b-a-p)^2)+b;
else
    if strcmp(req,'Ix') result=comp(part,'Ix');
    else
        if strcmp(req,'Iy') result=comp(part,'Iy');
        else
            if strcmp(req,'centX') result=comp(part,'centx')*IsPos;
            else
                if strcmp(req,'centY') result=comp(part,'centy');
                else
                    if strcmp(req,'comp')
                        result(1,1)=vertrap(b,Origh,a,p,'area');
                        result(1,2)=vertrap(b,Origh,a,p,'circ');
                        result(1,3)=vertrap(b,Origh,a,p,'centx');
                        result(1,4)=vertrap(b,Origh,a,p,'centy');
                        result(1,5)=vertrap(b,Origh,a,p,'ix');
                    else
                        if strcmp(req,'draw')
                            result=drawshape(part','Vertrap','Vertrap',
                                'Origin',h+Origh,0,0,'');
                        else
                            disp ('You tried to do something I don\'t know how to
')
                            disp ('do.
')
                            disp ('Wrong argument.
')
                            disp ('Sorry. I can\'t help you with that.
')
                            disp ('I can only do things I know how to do.
')
                        end
                    end
                end
            end
        end
    end
else
    if strcmp(req,'draw')
        disp ('You tried to do something I don\'t know how to
')
        disp ('do.
')
        disp ('Wrong argument.
')
        disp ('Sorry. I can\'t help you with that.
')
        disp ('I can only do things I know how to do.
')
    else
        disp ('You tried to do something I don\'t know how to
')
        disp ('do.
')
        disp ('Wrong argument.
')
        disp ('Sorry. I can\'t help you with that.
')
        disp ('I can only do things I know how to do.
')
    end
end

% End of file

% Matlab Library Function file

% reactions=[rad2xy([alpha mag(1) xcor ycor; beta mag(2) xcor ycor])];
return;
result(1,6)=vertrap(b,Origh,a,p,'iy');
elseif strcmp(req,'draw')
xcentroid=vertrap(b,Origh,a,p,'centx');
ycentroid=vertrap(b,Origh,a,p,'centy');
X=[0 Origh Origh 0 0];
Y=[0 p p+a b 0];
fill(X,Y,'r')
hold on;
ColA=strvcat('Area','Circumference','Centroid X','Centroid Y','Ix','Iy');
ColB=makecol(vertrap(b,Origh,a,p,'comp'));
plot(xcentroid,ycentroid,'ko',0,0,'g*')
hold off
axis ('equal')
titleblock(ColA,ColB)
expandaxis(5,5,5,0)
else
 disp('That is not a valid request, try area, circ, Ix, Iy, centX,')
disp('centY, comp.')
disp('You must use single quotes')
end
return;

B.35 vertria.m Matlab Library Function file

function [result]=vertria(b,h,p,req)
%VERTRIA Horizontal triangle shape routine.
% VERTRIA(BASE,HEIGHT,P,REQUEST)
%
% Shape described: Triangle with a vertical base.
%
% Datum location: Leftmost point of base.
%
% Input arguments:
% BASE: Length of base (base must be a horizontal line).
% HEIGHT: Vertical distance between base and vertex may be negative.
% P: Horizontal distance from the datum to lower edge of small edge. May
% be negative.
%
% Requests: (Must be in single quotes)
% 'area': Area of the shape.
% 'circ': Circumference of shape.
% 'Ix': Area moment of inertia about the neutral x axis.
% 'Iy': Area moment of inertia about the neutral y axis.
% 'centX': Distance from datum to centroid in the x direction.
% 'centY': Distance from datum to centroid in the y direction.
% 'comp': All of the above in a 1x6 matrix.
% 'draw': Show the shape graphically.
%
% User m functions called: hortria.m, titleblock.m makecol.m and expandaxis.m
%-----------------------------------------------
req=lower(req);
origH=h;
b=abs(b);
if b<0
 disp('Base must be a positive length')
return
end

if strcmp(req,'area') result=b*h/2;
elseif strcmp(req,'circ') result=sqrt(h^2+p^2)+sqrt(h^2+(b-p)^2)+b;
elseif strcmp(req,'Iy') result=b*h^3/36;
elseif strcmp(req,'Ix')
 if p==0 | p==b % right triangle
 result=b*h^3/36;
else
 if p<0 % vertex extends below base
 result=b*h^3/36;
end
end
midh = b * h / (abs(p) + b);
lsi = hortria(midh, p, h, 'ix');
lsa = hortria(midh, p, h, 'area');
lsd = hortria(midh, p, h, 'centy');
usi = hortria(midh, b, 0, 'ix');
usa = hortria(midh, b, 0, 'area');
usd = hortria(midh, b, 0, 'centy');
centroid = vertria(b, h, p, 'centy');
result = lsi + lsa * (centroid - lsd)^2 + usi + usa * (centroid - usd)^2;
end
if p < b % vertex extends above base
midh = b * h / p;
lsi = hortria(midh, -b, 0, 'ix');
lsa = hortria(midh, -b, 0, 'area');
lsd = hortria(midh, -b, 0, 'centy') + b;
usi = hortria(midh, p - b, h, 'ix');
usa = hortria(midh, p - b, h, 'area');
usd = hortria(midh, p - b, h, 'centy') + b;
centroid = vertria(b, h, p, 'centy');
result = lsi + lsa * (centroid - lsd)^2 + usi + usa * (centroid - usd)^2;
end
if p > 0 & p < b % vertex is across from the base
lsi = hortria(h, -p, 0, 'ix');
lsa = hortria(h, -p, 0, 'area');
lsd = hortria(h, -p, 0, 'centy') + p;
usi = hortria(h, b - p, 0, 'ix');
usa = hortria(h, b - p, 0, 'area');
usd = hortria(h, b - p, 0, 'centy') + p;
centroid = vertria(b, h, p, 'centy');
result = lsi + lsa * (centroid - lsd)^2 + usi + usa * (centroid - usd)^2;
end
elseif strcmp(req, 'centy')
if p < 0 % vertex extends below base
midh = b * h / (abs(p) + b);
lsa = hortria(midh, p, h, 'area');
lsd = hortria(midh, p, h, 'centy');
wa = vertria(b, h, p, 'area');
result = (lsa * lsd) + (usa * usd) / wa;
end
if p = 0 % right triangle
result = b / 3;
end
if p = b % right triangle
result = b * 2 / 3;
end
if p < b % vertex extends above base
midh = b * h / p;
lsa = hortria(midh, b, 0, 'area');
lsd = hortria(midh, b, 0, 'centy') + b;
wa = vertria(b, h, p, 'area');
result = (lsa * lsd) + (usa * usd) / wa;
end
if p > 0 & p < b % vertex is across from the base
lsa = hortria(h, -p, 0, 'area');
lsd = hortria(h, -p, 0, 'centy') + p;
wa = vertria(b, h, p, 'area');
result = (lsa * lsd) + (usa * usd) / wa;
end
elseif strcmp(req, 'centx')
result = origH / 3;
elseif strcmp(req, 'comp')
result(1,1) = vertria(b, origH, p, 'area');
result(1,2) = vertria(b, origH, p, 'circ');
result(1,3)=vertria(b,origH,p,'centx');
result(1,4)=vertria(b,origH,p,'centy');
result(1,5)=vertria(b,origH,p,'ix');
result(1,6)=vertria(b,origH,p,'iy');

elseif strcmp(req,'draw')
	xcentroid=vertria(b,origH,p,'centx');
ycentroid=vertria(b,origH,p,'centy');
X=[0 origH 0 0];
Y=[0 p b 0];
fill(X,Y,'r')
hold on;
ColA=strvcat('Area','Circumference','Centroid X','Centroid Y','Ix','Iy');
ColB=makecol(vertria(b,origH,p,'comp')');
plot (xcentroid,ycentroid,'ko',0,0,'g*')
hold off
axis ('equal')
titleblock(ColA,ColB)
extendaxis(5,5,5,0)
else
disp ('That is not a valid request, try area, circ, Ix, Iy, centX,')
disp ('centY, comp.')
disp ('You must use single quotes')
end
return;

B.36 xy2deg.m Matlab Library Function file

function [vector]=xy2deg(inputs)
% XY2DEG Converts vectors in standard form to degree angle form.
% XY2DEG (INPUTS) Routine takes any number of force vectors in standard form
% and returns them in degree angle format.
% % INPUTS: [XMAG,YMAG,XCOR,YCOR]
% % OUTPUTS: [ANGLE,MAG,XCOR,YCOR]
% % User m functions called: RD.m
%-----------------------------------------------------------------
[r,c]=size(inputs);

if c==4
	ycor=inputs(:,4); % pull coordinates out of input matrix
xcor=inputs(:,3); % pull coordinates out of input matrix
end

if c==2 % if coordinates were not given
xcor=zeros(r,1); ycor=zeros(r,1); % coordinates default to origin
end

mag=sqrt(inputs(:,1).^2+inputs(:,2).^2); % magnitudes from inputs
angle=RD(atan2(inputs(:,2),inputs(:,1)));

vector=[angle,mag,xcor,ycor]; % reassemble complete answer
return;
REFERENCES