The University of Glasgow

Department of Mechanical Engineering

Design and Implementation of a Composite Material Software

Project Report

Student Name: Tukur Ahmed Gidado

IAESTE Summer Trainee

Date: 15 August 2008

Table of Contents

Abstract

The aim of this project is to develop a small in-house version of a program to calculate the stiffness and strength of composite materials. The program should be able calculate the mechanical properties of unidirectional continuous fibre reinforced composite materials using the micromechanical approach. It includes a data management system which is connected to a database and lets a user choose a material for a fibre and a matrix to make a composite. The program should plot the properties against the volume fraction of the fibre in the composite. The program should compare the results with experimental values available from standard literature. The program was developed with MATLAB, in conjunction with GUIDE.

Introduction

Composite materials are materials engineered from two or more constituent materials with significant difference in physical properties from the constituents. The physical properties of composite materials can be estimated using various approaches. The easiest approach is the micromechanical approach which constitutes empirical models and models developed by different scientist over time. The micromechanical model may be far from the actual properties of the material due to various assumptions made to simplify calculations. For practical applications, it is necessary to compare results of empirical models with experimental data before making decisions for a composite material.

There are various composite programs in the market but are largely based on models and users have no means of verifying the accuracy of a model based on these programs. It will be desirable to have a program with a database of experimental values of different composite materials to compare against results calculated from models before selecting a material for a specific application.

This project code named compositor is to develop a small in-house version of a program to calculate the stiffness and strength of composite materials based on micromechanical models and compare the results with experimental values to ascertain a margin of error. The program was developed as a graphical user interface with MATLAB and GUIDE (Graphical User Interface Development Environment). The program is connected to a database of materials with mechanical properties of fibre, matrix and experimental data of mechanical properties of composite materials. The program interacts with the database to get data for the properties of fibre and matrix, calculates the stiffness of the composite material, and compares with the experimental values of the composite in a graph. The program also tells a user of the possible sources of variation and gives suggestions on possible fixes to the error.

Two versions of the program were implemented in this project. This report details the program as a whole in version one and goes on to discuss the changes and improvements made to the program in version 2.

Methodology

The project has two main phases;

- The development of a GUI in MATLAB's GUIDE and the code in a MATLAB "m" file to support the GUI.
- The development of a material database and searching literature for experimental values of mechanical properties of composites.

Version 1

Graphical User Interface (GUI)

The GUI was designed using with MATLAB's GUIDE with three tabbed panels. GUIDE does not support the undocumented UITAB function and an alternate approach was designed with toggle buttons and panels. Each panel is associated with a toggle button which sets the visibility property of its respective panel to "on" and of other panels to "off" thus displaying or hiding the panel.

The main program window has three tabs; The Data Management tab, The Database Viewer tab, and The Micromechanics tab.

The Database Management tab

The database management tab activates the database management panel which is divided into three panels;

- **The Fibre Data panel**: This panel consists of editable textboxes with static text aligned to the left of each. The static text is the required material property which will be entered into the respective editable textbox. The panel also includes a save button which a user will click to save the data he has entered in the editable textboxes to a database.
- **The Matrix Data panel** : same as above.
- **Experimental Data panel**: same as above.

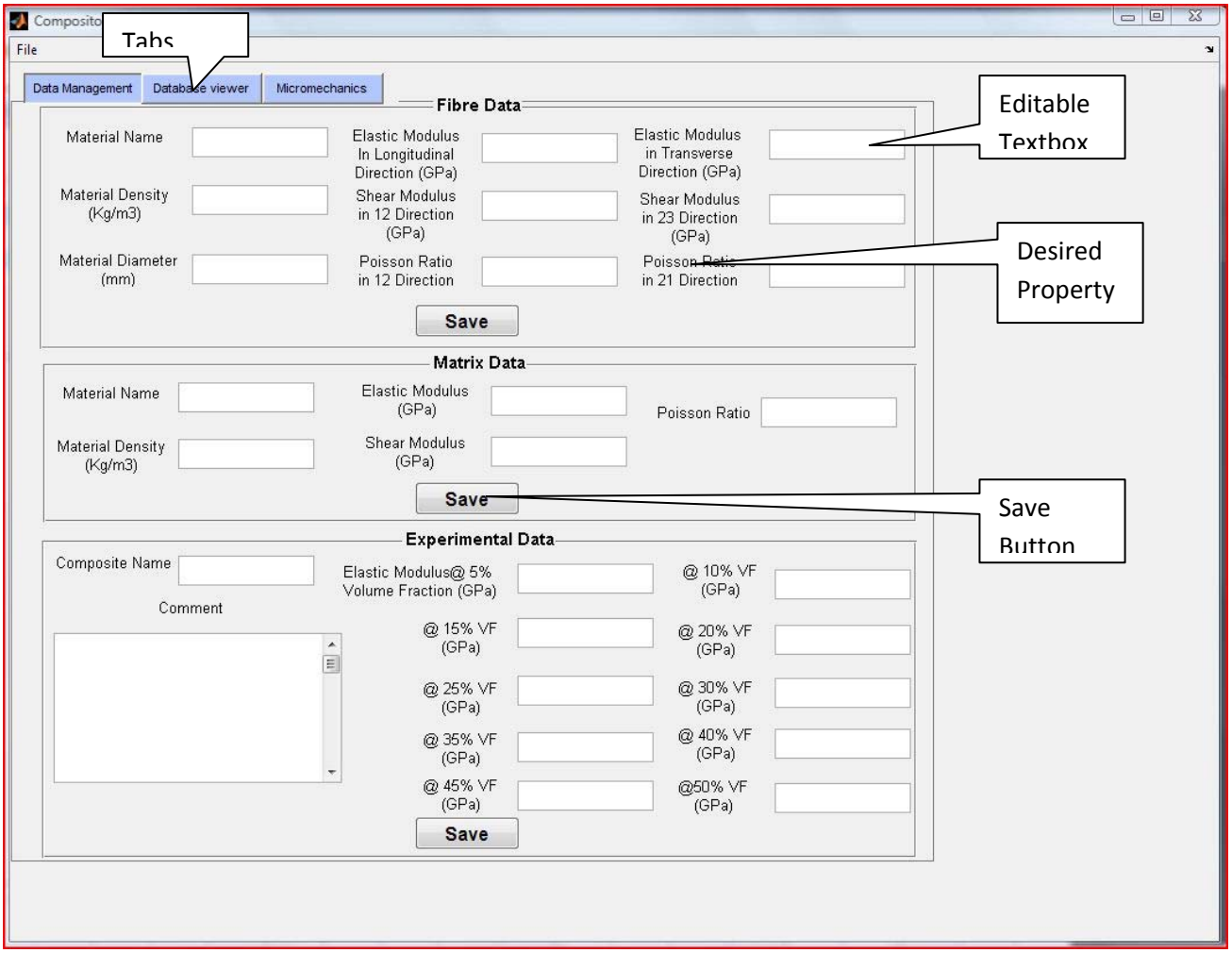


Figure 1… The Data Management Panel

The Database Viewer tab

The database viewer tab activates the database management panel which has three tables;

- **The Fibre Data table**: This table is a simple two-dimensional table that will read the properties of fibre materials from the database and display them on the GUI for user to see the list of available materials and properties straight from the GUI window.
- **The Matrix Data table:** This table is for matrix materials.
- **Experimental Data table:** This table is for experimental data values.

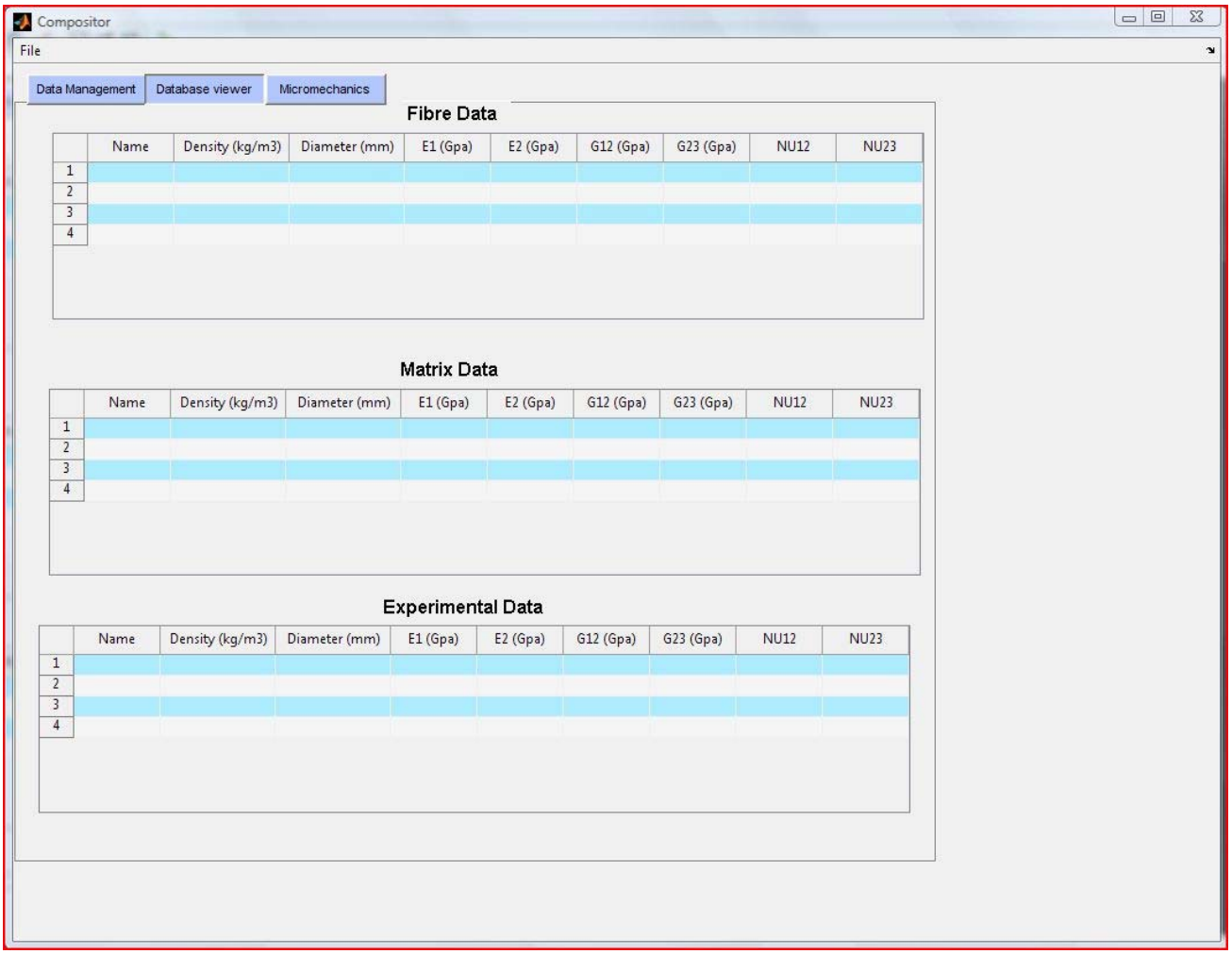


Figure 2. The Data viewer Panel

The Micromechanics tab

The micromechanics tab activates the micromechanics panel which is made up of the following sections or panels;

- **The Material selection Panel:** The material selection panel on the upper left of the GUI has two editable textboxes and an "OK" pushbutton. A user inputs the name of a fibre and matrix in their respective textboxes, and click on the push button.
- **The Model Selection Panel:** This panel is further subdivided into three
    - *The Property selection radio button group:* the user uses this to select the property he wants to investigate.

- *The Plot toggle button group;* the user toggles this group of buttons up and down to select a model he wants to plot on the adjacent graph axes.
- *The normalize plot button group;* the user toggles any of these buttons to normalize and plot in the adjacent graph axes.

Plot Axes: Two plot axes are included in this panel of the GUI to dynamically plot graphs relating to the users selection.

Comment Box: The comment box located at the lower left of the panel is a static textbox to display results, comments and references associated with the current material selections.

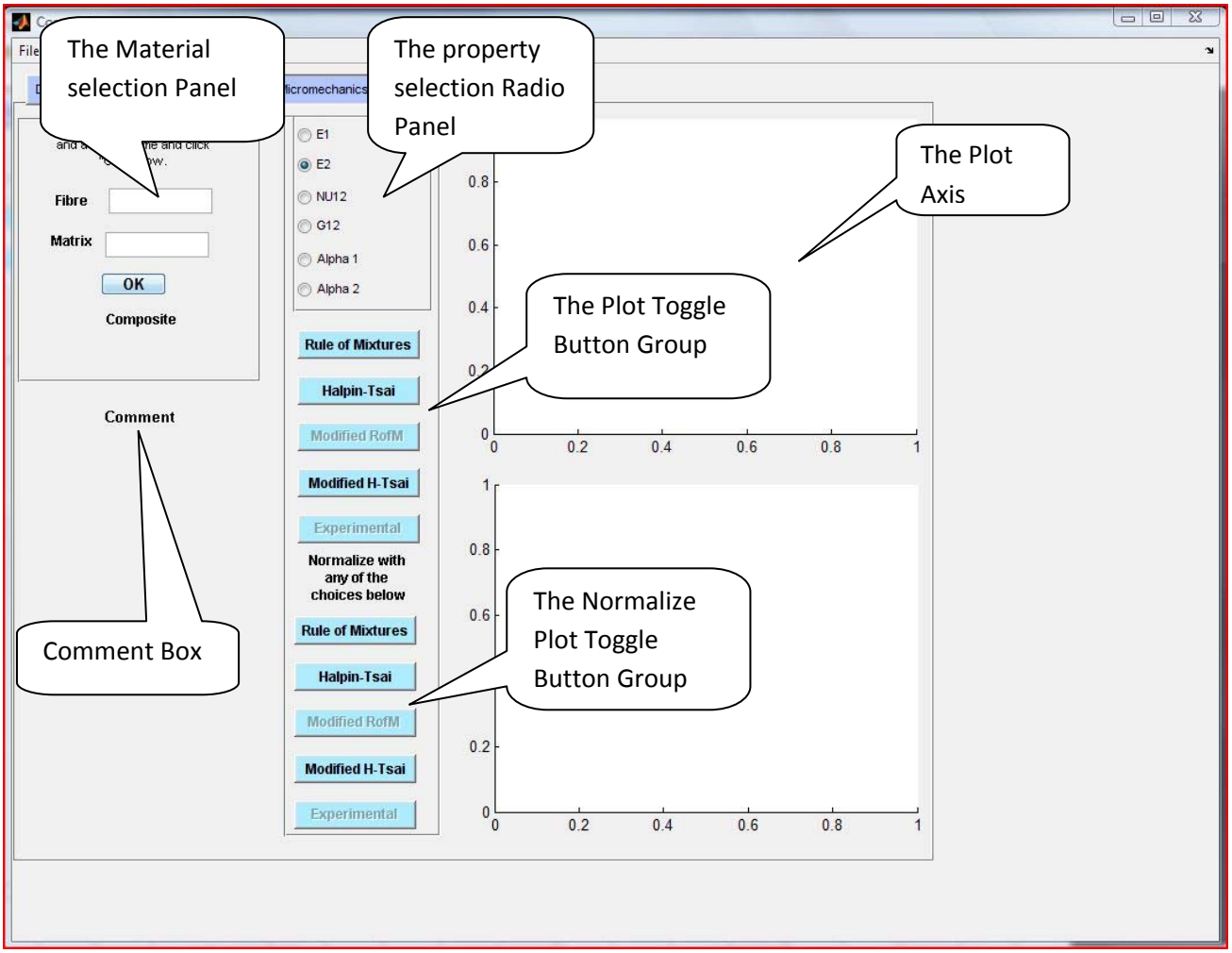


Figure 3. The Micromechanical Panel

The M-file (code)

The GUI is driven by an m-file of the same file name behind the scenes to achieve the results. The GUI has a single M-file with all the codes for the GUI except the micromechanical models. The micromechanical models were written as MATLAB functions in separate M-files each. All codes necessary for the execution of the program are attached as appendix to this report.

Database

A database was initially developed in Microsoft Access to contain all the materials data. The database was linked to MATLAB via a commercial ODBC (Open Database Connectivity) driver. The requirement of a commercial driver to connect to the database increased complexity to the program and a new database was developed in Microsoft Excel with sheets as tables. The time required to read and write data to excel files was too long and a database was developed in MATLAB using MATLAB's MAT file format and structures to store data. This MAT file mimics a database by saving data into different structures as an alternative to variables.

The database has three structures; The Fibre, Matrix and Experimental structures.

The fibre structure has fields with the name and mechanical properties of the material and stores data as numeric values directly under the fields.

The matrix structure has fields with the name and mechanical properties of the material and stores data as numeric values directly under the fields.

The Experimental structure has nested structures with the names of materials as fields. The fields themselves are structures with subfields as property names. The data in the experimental database is saved in pairs with the value of a property saved with the corresponding volume fraction of fibre in the composite material. References to the experimental values as well as comments are saved under the material name structure.

Program Implementation

Database Management

The database management system interacts with MATLAB functions implemented in the M-file. The system has a search algorithm using MATLAB's "strcmpi" function in a loop, to search for a material in the database according to the user input; the "strcmpi" function is case independent so the user can enter small or big case letters. The search algorithm also uses the "strtrim" function to ignore all white spaces in a material name. This algorithm has a limitation in that the user has to know the exact name of the material before using it. If the material the user enters is not in the database, a new material dialog box pops out and asks if the user wants to create a new material entry. If he clicks on yes a new entry is added to the end of the database ready for the user to input mechanical property values.

Figure 4. The Create new Entry Dialog Box

The database management system also has a save algorithm for writing to the database when changes have been made. The save algorithm has two features;

- To save any changes to a workspace variable within the workspace without writing to the database.
- To write only changed variables to the database when the save button is clicked.

Database Viewer

The database viewer lets the user have a look at the materials in the database before searching for a material to use. The database viewer utilizes MATLAB's UITABLE functions and

properties. When the program loads, all data from the database is saved to MATLAB's handle structure. Structured data is converted into cell arrays with MATLAB's "struct2cell" function and saved as workspace variables. The UITABLE's "CellEditCallback" extracts the data required for display from the workspace and displays them in the tables in the GUI.

Micromechanics

User Input Panel: This is the panel where a user inputs the fibre and matrix name. It has a search algorithm very similar to the one in the database management. After searching for a fibre and matrix and if they exist in the database, the data associated with that material is extracted from the database and converted into numeric data and saved to workspace variables for later use. A user then clicks on the "OK" button, the "OK" button's callback uses MATLAB's "strcat" function to make a composite name consisting of the fibre name, an underscore character and the matrix name in that order. If a fibre and/or matrix does not exist in the database a message box pops out telling a user to add the material to the database from the database management tab.

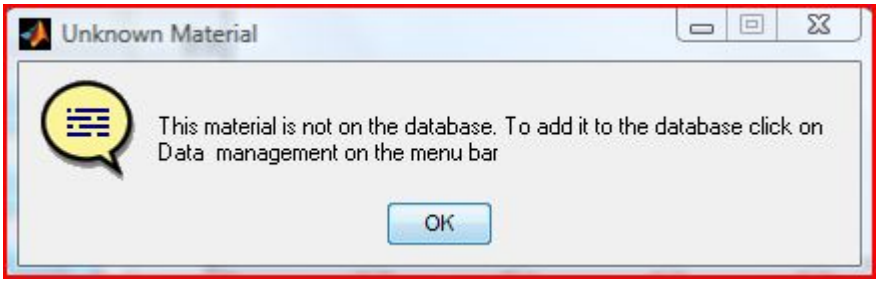


Figure 5… The Unknown Material Message Box

Thereafter the "OK" button's callback calls the search algorithm to search for experimental data related to the composite material. If the experimental data exists for that composite, the data is saved as variables to the GUI's handles structure for later use.

The Radio Button Panel: The radio button panel lets a user choose which mechanical property he wants to investigate. It has mutually exclusive radio buttons (i.e. a user can choose only one property at a time) grouped into a button group. The mutual exclusivity routine is implemented in the button group's "SelectionChange" callback function. If a user selects a property, the button group below activates the buttons associated with that property by making the button's "enable" property on and other buttons enable property "off".

The Plot Button Group: The plot button group is a group of toggle buttons used to plot the graph of the selected model on the adjacent graph axis. Toggle buttons are used to avoid clutter on the graph axis, it has an up and down state for on and off respectively. When the button is clicked once the graph is plotted, when it is clicked again the graph is deleted. An algorithm using a switch loop is used to determine the state of the button and the necessary action executed accordingly. When the button is on it returns a value of 1 whereas, when it is off returns 0. These two values are switched in a loop to execute the code associated with each state. The toggle button's callback gets all the data from the workspace and calls the appropriate micromechanical model function to do the calculation and return the results for plotting. With the exception of the experimental button which reads the data directly from the workspace and plots it. If experimental data is not available for that composite material, a user is notified via a message box.

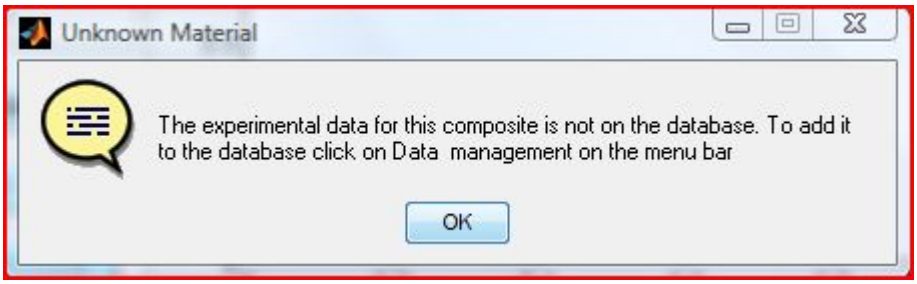


Figure 6… Unknown Material Message box for experimental values

The Normalized Plot Button Group: This button group is very similar to the button group above but with the addition of another function for normalizing. The normalize function divides the values of the results from calculations of other models by the results of the model clicked before plotting into the adjacent graph axis. With the exception of the experimental button, the experimental button calls a loop to extract a matching number of data points from other results to the number of data points in the experimental data. Afterward normalizes and plots onto the graph axis.

The Comment Box: The comment box is activated only when the experimental button is clicked it reads the comments and or references associated with a particular set of experimental values and displays them on the screen using the handles of a static textbox.

The Plot Axes: The graphs are plotted in the axes on the GUI. The axes are programmed to adjust their limits according to the data plotted and have features to differentiate the colors of the lines plotted and a legend to serve as a key to the plots.

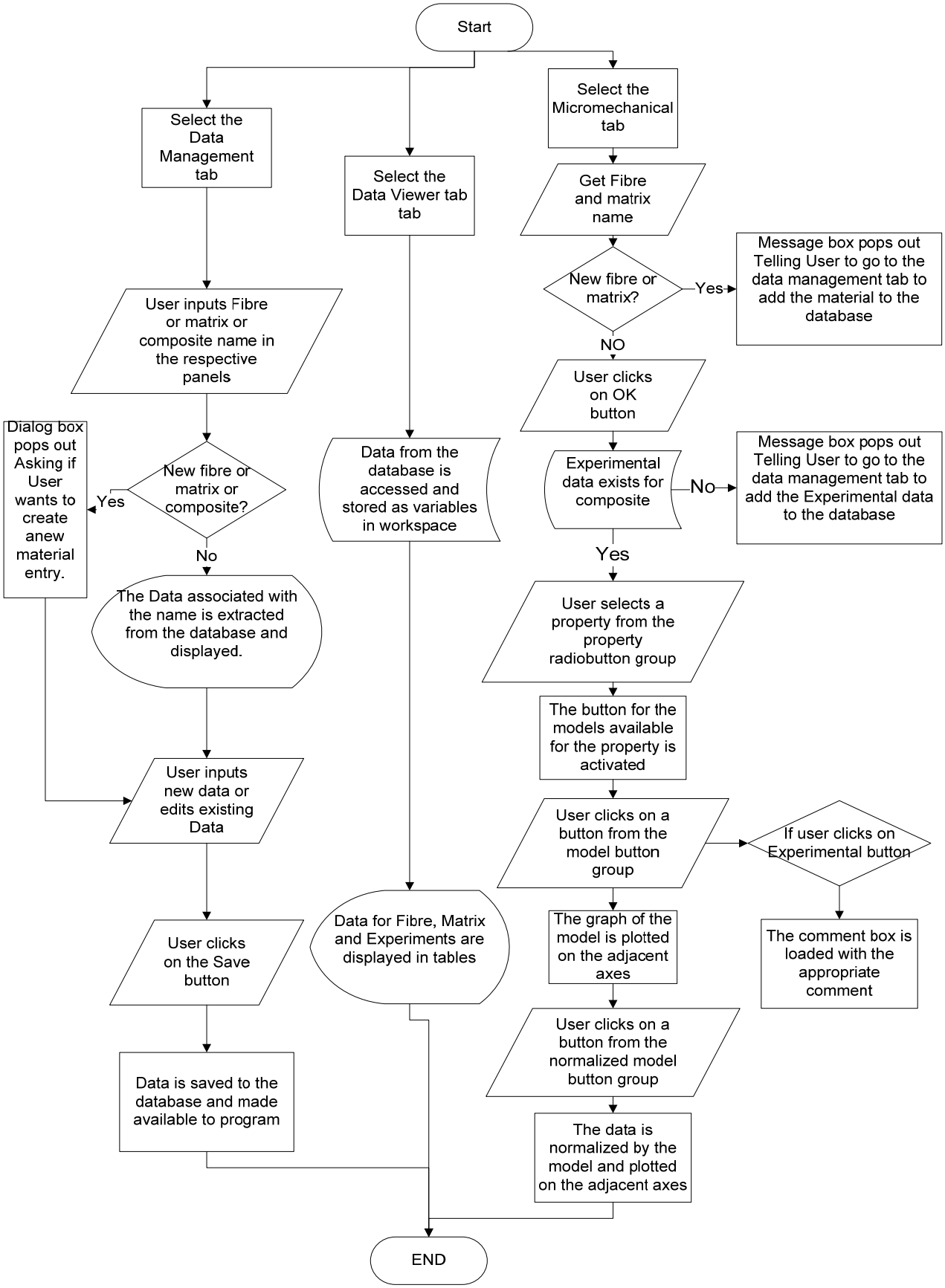


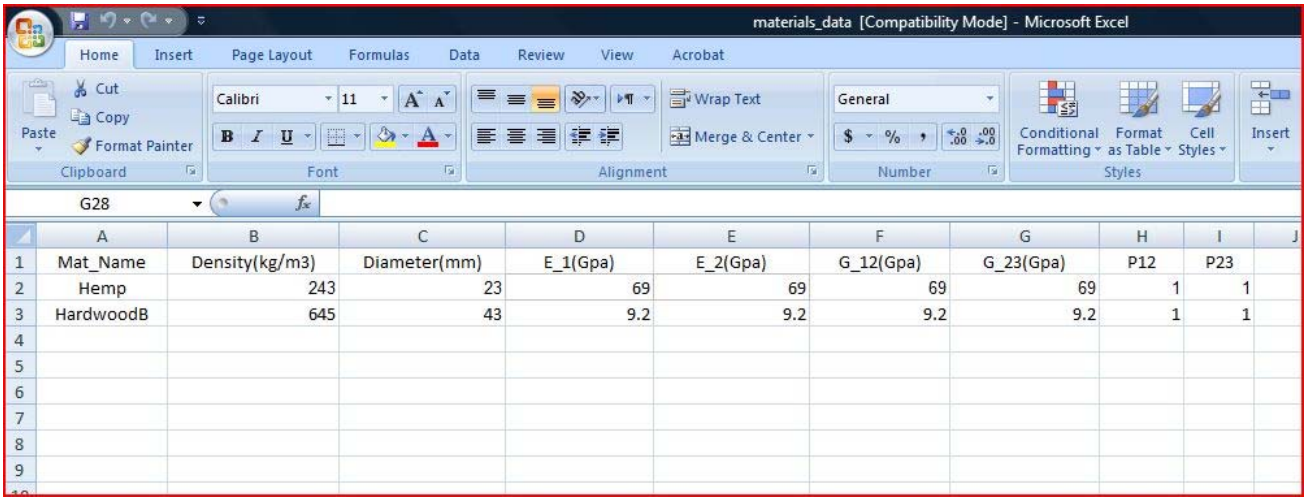
Figure 7… Flow chart for program operation

Version 2

A major modification has been made to version 1 of the program and implemented in a newer version designated as version 2. The Database viewer and Database management panels have been removed and a new panel for database management implemented. The database was changed. The micromechanical panel was not changed but a few bugs were fixed from the version 1.

Database

The new database was built in Microsoft excel. A single database was implemented in a single excel workbook with the workbook sheets names as Fibres, Matrices and Experimental containing fibres, matrices and experimental data respectively. In each database the names of the columns are the material properties while the names of the rows are the material name.



	A	B	C	D	E	F	G	H	I
1	Mat_Name	Density(kg/m3)	Diameter(mm)	E_1(Gpa)	E_2(Gpa)	G_12(Gpa)	G_23(Gpa)	P12	P23
2	Hemp	243	23	69	69	69	69	1	1
3	HardwoodB	645	43	9.2	9.2	9.2	9.2	1	1
4									
5									
6									
7									
8									
9									

Figure 8… Screen shot of the new database

The new database is not read into MATLAB via Excel Active Server as was the first attempt to implement a database in Excel. This database simply uses MATLAB's "xlsread" and "xlswrite" functions which are functions specifically designed to read and write excel files.

GUI

The GUI has been changed to only two tabbed panels; one for database management and the other for micromechanics. The micromechanics have not changed from the version 1. The new database management tab opens a database management panel that serves as an editor and viewer for materials contained in the database.

The database management panel uses MATLAB GUIDE's newly (New in MATLAB r2008a) included UITABLE control to get data from the excel file and display them on the screen. The raw data is extracted from excel with the xlsread function and saved to the UITABLE's handle structure. The data is then displayed on the screen through the generic set function. If any modifications were made to data in the GUI, the changes were retrieved through the generic get function and saved to the handles structure. Editing of data in the database was implemented with the UITABLE's "celleditcallback" function. Whenever changes are made to data in a cell of the table and a user moves away from the cell, the celleditcallback is called upon to get the changes and save it to the handles structure. However changes made in the GUI are not saved to the database automatically, a user is required to click on the save button. The save button uses the xlswrite function to permanently write data on the database.

The data extracted from the database cannot be used for calculation and because it is in the cell format. The cell2mat function is always called to convert data from cell format to matrix format.

If a user wants to add a new material to the database, he can click on a newly added new button. When the new button is clicked, the button's callback function is called. The callback function searches the end of the database and creates a new row of cell arrays at the end. The callback also initializes the cells that will contain strings with a character array. The user can then input the material properties in the cells provided and click on the save button. If the user does not click on the save button, the new row is not saved and the next time the program runs it will not display the row.

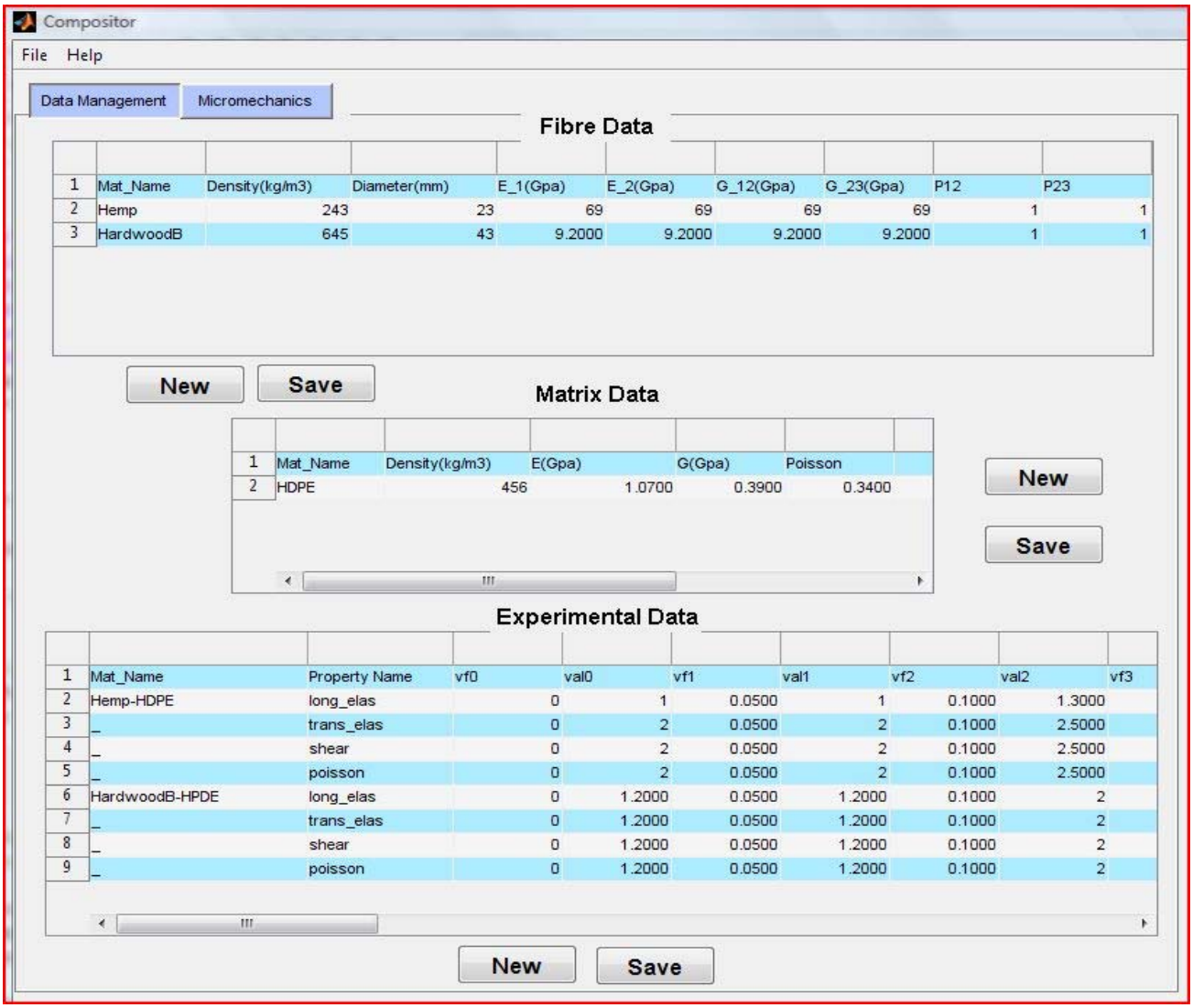


Figure 9… The new Database management Panel

Deployment

For the software to be useful, it has to be deployed from the MATLAB development environment and packaged as a platform independent application so that different users can use it without having MATLAB. The Deployment is done with MATLAB's deploytool which is a GUI based tool for compiling and packaging applications. The deploytool can be called from the command window with the command "deploytool". A new window pops out with deployment settings. An option of deploying with a complete set of MATLAB's runtime libraries known as MATLAB Compiler Runtime (MCR) can be selected. This option is for users without MATLAB installed on their systems or users with a different MATLAB version from the version of the compiled application. A c language compiler must be installed on the system before nay compilation. MATLAB comes with an in-built c-compiler LCC. But one has to setup the compiler

with the command mcc –setup and choose the compiler one will use to build the application. If the application successfully builds without any errors, one has to package it by clicking the package button. One can add all the necessary files required with the application to the package and create a ZIP package.

Constraints:

The MATLAB program is a powerful function packed High Level Programming language, however the graphic user interface development environment of MATLAB was designed to support very basic GUI development and hindered the ability to develop a High level GUI. MATLAB's inbuilt function for searching is very limited and a user has to implement his search algorithm for better functionality.

Database programs are not easy to connect to MATLAB and the use of MATLAB's MAT file and structures to mimic a database is far from an actual database.

Conclusions

A micromechanics model is used to predict the mechanical properties of a unidirectional fibre reinforced composite material with some degree of accuracy. We cannot ascertain the degree of accuracy of the model and predict how these materials will behave if deployed in a practical application. Experimental values are much more realistic of the actual mechanical properties of the composites. However, experimental values are subjected to the conditions of experiments and the manufacturing method of the composite material.

By developing a software to compare the prediction of models against the results of experiments,

We can determine the margin of error and decide which model or values will best apply to our application.

The software can be extended to cover laminate theory and predict more mechanical properties and compare with experimental data, giving designers a reliable tool in the selection of composite materials for practical applications.

Future developments

Future developments on the program should include

- A Laminate section that will implement the macro mechanical (Laminate theory) approach to calculating composite materials mechanical property.
- The program should have a more efficient and flexible data that lets a user input a wider range of composites data.
- The program should link to various literature databases to extract experimental results to compare against.
- An intelligent approach to material selection by suggesting to the user what materials are available.
- A 3-D section to extend the program beyond unidirectional continuous fibre reinforced composites.

APPENDICES

APPENDIX A

MATLAB Code

```
function varargout = Compositor(varargin)
% Matlab generated code for generating the figure file or GUI
% Begin initialization code - DO NOT EDIT

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Compositor_OpeningFcn, ...
                   'gui_OutputFcn',  @Compositor_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

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


% --- Executes just before Compositor is made visible.
function Compositor_OpeningFcn(hObject, eventdata, handles, varargin)
%All required variable files are to loaded unto workspace before
%GUI is made visible
handles.output = hObject;


% read the required variables from excel worksheet and store in
%the handle structure for universal acess from all functions

[n,t, fibre_raw_data] = xlsread('materials_data.xls', 'Fibres');
[n,t, matrix_raw_data] = xlsread('materials_data.xls', 'Matrices');
[n,t, expr_raw_data] = xlsread('materials_data.xls', 'Experimental');

handles.Fibres = fibre_raw_data;
handles.Matrices = matrix_raw_data;
handles.Experimental = expr_raw_data;

%set the handles of the table to the excel data to display it on screen

set(handles.fibre_data_table,'Data',fibre_raw_data);
set(handles.matrix_data_table,'Data',matrix_raw_data);
set(handles.experimental_data_table,'Data',expr_raw_data);

%at initialization set the micromechanics panel should to invisible
```

```
%and the data management panel to visible

   set(handles.data_manage_panel,'visible','on');
   set(handles.micro_panel,'visible','off');

% set all the toggle buttons on the micromechanical panel to disable
%they are enabled after the radio button for property is selected

         set(handles.r_of_m,'enable','off');
         set(handles.H_tsai,'enable','off');
         set(handles.Expr,'enable','off');
         set(handles.modhtsai,'enable','off');
         set(handles.modr_of_m,'enable','off');
         set(handles.nh_tsai,'enable','off');
         set(handles.nr_of_m,'enable','off');
         set(handles.nmodr_of_m,'enable','off');
         set(handles.nmodh_tsai,'enable','off');
         set(handles.nexpr,'enable','off');
         set(handles.schap,'Visible','off');
         set(handles.nschap,'visible','off');

% reference the selectionchange functions of the button group and
% radio button group here at the opening function

set(handles.button_group,'SelectionChangeFcn',@button_group_SelectionChangeFcn);
set(handles.radiobutton_group,'SelectionChangeFcn',@radiobutton_group_SelectionChangeFcn);
% Update handles structure
guidata(hObject, handles);

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


% --- Outputs from this function are returned to the command line.
function varargout = Compositor_OutputFcn(hObject, eventdata, handles)
% Get default command line output from handles structure
varargout{1} = handles.output;

%The tab functionality function, a workaround to tabbing
function button_group_SelectionChangeFcn(hObject,eventdata)
%retrieve GUI data, i.e. the handles structure
handles = guidata(hObject);
%the tag of the selected toggle button is checked and the corresponding
%panel is set to visible while the other to invisible
switch get(eventdata.NewValue,'Tag') % Get Tag of selected object.
    case 'data_manage_tgl'
         set(handles.data_manage_panel,'visible','on');
          set(handles.micro_panel,'visible','off');
         guidata(hObject, handles);

   case 'micro_tgl'
     set(handles.micro_panel,'visible','on');
     set(handles.data_manage_panel,'visible','off');
      guidata(hObject, handles);

end
```

```
% --- Executes when selected object is changed in radiobutton_group.
function radiobutton_group_SelectionChangeFcn(hObject, eventdata, handles)
%retrieve GUI data, i.e. the handles structure
handles = guidata(hObject);

%rbcontrol is a variable to output the selected radio button to other
%functions for use
%e1_rb is the E1 radio button on the micromechanics panel
%e2_rb is the E2 radio button on the micromechanics panel
%v12_rb is the NU12 radio button on the micromechanics panel
%g12_rb is the G12 radio button on the micromechanics panel
%alp1_rb is the Alpha 1 radio button on the micromechanics panel
%alp2_rb is the Alpha 2 radio button on the micromechanics panel

global rbcontrol;
switch get(eventdata.NewValue,'Tag') % Get Tag of selected object.
   case 'e1_rb'
         set(handles.r_of_m,'enable','on');
         set(handles.H_tsai,'enable','off');
         set(handles.Expr,'enable','on');
         set(handles.modhtsai,'enable','off');
         set(handles.modr_of_m,'enable','off');
         set(handles.nh_tsai,'enable','off');
         set(handles.nr_of_m,'enable','on');
         set(handles.nmodr_of_m,'enable','off');
         set(handles.nmodh_tsai,'enable','off');
         set(handles.nexpr,'enable','on');
          set(handles.schap,'Visible','off');
         set(handles.nschap,'visible','off');
%cla clears the axis i.e deletes the graph
         cla(handles.plotter)
         cla(handles.plotass)
          set(handles.comment,'string','')
            set(handles.reference,'string','')
         rbcontrol = 1;
         guidata(hObject, handles);

    case 'e2_rb'

         set(handles.r_of_m,'enable','on');
         set(handles.H_tsai,'enable','on');
         set(handles.Expr,'enable','on');
         set(handles.modhtsai,'enable','on');
         set(handles.modr_of_m,'enable','off');
         set(handles.nh_tsai,'enable','on');
         set(handles.nr_of_m,'enable','on');
         set(handles.nmodr_of_m,'enable','off');
         set(handles.nmodh_tsai,'enable','on');
         set(handles.nexpr,'enable','on');
          set(handles.schap,'Visible','off');
         set(handles.nschap,'visible','off');
          cla(handles.plotter)
         cla(handles.plotass)
          set(handles.comment,'string','')
            set(handles.reference,'string','')
```

```
        rbcontrol = 2;
        guidata(hObject, handles);

    case 'v12_rb'
          set(handles.r_of_m,'enable','on');
        set(handles.H_tsai,'enable','off');
        set(handles.Expr,'enable','on');
        set(handles.modhtsai,'enable','off');
        set(handles.modr_of_m,'enable','off');
        set(handles.nh_tsai,'enable','off');
        set(handles.nr_of_m,'enable','on');
        set(handles.nmodr_of_m,'enable','off');
        set(handles.nmodh_tsai,'enable','off');
        set(handles.nexpr,'enable','on');
         set(handles.schap,'Visible','off');
        set(handles.nschap,'visible','off');
         cla(handles.plotter)
        cla(handles.plotass)
         set(handles.comment,'string','')
           set(handles.reference,'string','')
        rbcontrol = 3;
        guidata(hObject, handles);

    case 'g12_rb'
         set(handles.r_of_m,'enable','on');
        set(handles.H_tsai,'enable','on');
        set(handles.Expr,'enable','on');
        set(handles.modhtsai,'enable','off');
        set(handles.modr_of_m,'enable','on');
        set(handles.nh_tsai,'enable','on');
        set(handles.nr_of_m,'enable','on');
        set(handles.nmodr_of_m,'enable','on');
        set(handles.nmodh_tsai,'enable','off');
        set(handles.nexpr,'enable','on');
         set(handles.schap,'Visible','off');
        set(handles.nschap,'visible','off');
         cla(handles.plotter)
        cla(handles.plotass)
         set(handles.comment,'string','')
           set(handles.reference,'string','')
        rbcontrol = 4;
        guidata(hObject, handles);

    case 'alp1_rb'
       set(handles.r_of_m,'enable','off');
        set(handles.H_tsai,'enable','off');
        set(handles.Expr,'enable','off');
        set(handles.modhtsai,'enable','off');
        set(handles.modr_of_m,'enable','off');
        set(handles.nh_tsai,'enable','off');
        set(handles.nr_of_m,'enable','off');
        set(handles.nmodr_of_m,'enable','off');
        set(handles.nmodh_tsai,'enable','off');
        set(handles.nexpr,'enable','off');
         set(handles.schap,'Visible','on');
        set(handles.nschap,'visible','on');
         cla(handles.plotter)
```

```
        cla(handles.plotass)
         set(handles.comment,'string','')
           set(handles.reference,'string','')
        rbcontrol = 5;
        guidata(hObject, handles);

    case 'alp2_rb'
          set(handles.r_of_m,'enable','on');
        set(handles.H_tsai,'enable','off');
        set(handles.Expr,'enable','off');
        set(handles.modhtsai,'enable','off');
        set(handles.modr_of_m,'enable','off');
        set(handles.nh_tsai,'enable','off');
        set(handles.nr_of_m,'enable','on');
        set(handles.nmodr_of_m,'enable','off');
        set(handles.nmodh_tsai,'enable','off');
        set(handles.nexpr,'enable','off');
         set(handles.schap,'Visible','on');
        set(handles.nschap,'visible','on');
         cla(handles.plotter)
        cla(handles.plotass)
         set(handles.comment,'string','')
           set(handles.reference,'string','')
        rbcontrol = 6;
        guidata(hObject, handles);

end


% --- Executes on button press in r_of_m.
function r_of_m_Callback(hObject, eventdata, handles)

%get the name of the fibre and save it to the variable current_mat
%if the fibre box is empty just return

current_mat = get(handles.fibre,'string');
if isempty(current_mat)
   return
end
%get the on/off state of the toggle button and save it to val
val = get(hObject,'Value');

% declare global varables required for calculation
% e1f is the elastic modulus of the fibre in the 1 direction
%e2f is the elastic modulus of the fibre in the 2 direction
%em is the elastic modulus of the matrix
%g12f is the shear modulus of the fibre
%gm is the shear modulus of the matrix
% nu12f is the poisson ratio of the fibre
%num is the poisson ratio of the matrix
%therofm is the calculated value of the rule of mixtures from this
%function saved to a global variables for use in other functions
%rbcontrol is a variable to check which mechanical property has been
%selected and call the neccessary function for calculation accordingly.

global e1f e2f em g12f gm nu12f num therofm rbcontrol;
```

```
%vf is a volume fraction vector with 101 points
vf = 0:0.005:0.50;
if rbcontrol == 1 %if E1 is selected
%call the rule of mixture in parallel direction function E1 and solve for
%E1 and return a vector of 101 values matching the volume fraction
therofm = E1(vf,e1f,em);
% Switch the state of the toggle button on/off
switch val
%case 1, when the state of the toggle button is on
   case 1
%plot the rule of mixture against volume fraction on the first axis
axes(handles.plotter);
plot(vf,therofm,'b-')
xlabel('volume fraction')
ylabel('Elastic Moduli (Gpa)')
grid on
hold on;
%case 0, when the state of the toggle button is off
   case 0
      axes(handles.plotter);
% A work around to clearing the axis is to plot it in white color
   plot(vf,therofm,'w-')
end
end
if rbcontrol == 2 %if E2 is selected
therofm = E2rofm(vf,e2f,em); %the rule of mixture in parallel direction
% Switch the state of the toggle button on/off
switch val
%case 1, when the state of the toggle button is on
   case 1
%plot the rule of mixture against volume fraction on the first axis
axes(handles.plotter);
plot(vf,therofm,'b-')
xlabel('volume fraction')
ylabel('Elastic Moduli (Gpa)')
grid on
hold on;
%case 0, when the state of the toggle button is off
   case 0
      axes(handles.plotter);
   plot(vf,therofm,'w-')
end
end
if rbcontrol == 3 %if NU12 is selected
therofm = NU12(vf,nu12f,num); %the rule of mixture in parallel direction
% Switch the state of the toggle button on/off
switch val
%case 1, when the state of the toggle button is on
   case 1
%plot the rule of mixture against volume fraction on the first axis
axes(handles.plotter);
plot(vf,therofm,'b-')
xlabel('volume fraction')
ylabel('Elastic Moduli (Gpa)')
grid on
hold on;
%case 0, when the state of the toggle button is off
   case 0
```

```
      axes(handles.plotter);
   plot(vf,therofm,'w-')
end
end
if rbcontrol == 4 %if G12 is selected
therofm = G12rofm(vf,g12f,gm); %the rule of mixture in parallel direction
% Switch the state of the toggle button on/off
switch val
%case 1, when the state of the toggle button is on
   case 1
%plot the rule of mixture against volume fraction on the first axis
axes(handles.plotter);
plot(vf,therofm,'b-')
xlabel('volume fraction')
ylabel('Elastic Moduli (Gpa)')
grid on
hold on;
%case 0, when the state of the toggle button is off
   case 0
      axes(handles.plotter);
   plot(vf,therofm,'w-')
end
end


% --- Executes on button press in normalized rule of mixture button.
function nr_of_m_Callback(hObject, eventdata, handles)

%get the name of the fibre and save it to the variable current_mat
%if the fibre box is empty just return

current_mat = get(handles.fibre,'string');
if isempty(current_mat)
   return
end
%get the on/off state of the toggle button and save it to val
val = get(hObject,'Value');

% declare global varables required for calculation within this function
%themrofm is the results from the modified rule of mixtures
%themhtsai is the result from the modified halpin-Tsai
%expr_vf is the experimental volume fraction from the database
%exprofm are values extracted from the results of the rule of mixtures
% to match the size of expr_val so they can divide each other.

global therofm rbcontrol thehtsai expr_vf
global themrofm themhtsai expr_val exprofm;
vf = 0:0.005:0.50;  %a volume fraction vector
if rbcontrol == 1
%normalisation divide results of other models by the rules of mixture
thenrofm = therofm./therofm;
thenexpr = expr_val./exprofm;
switch val
%case 1, when the state of the toggle button is on
   case 1
%plot the normalised rule of mixture against
%volume fraction on the second axis
```

```
axes(handles.plotass);
plot(vf,thenrofm,'b-')
ylim([-1 3]);
xlabel('volume fraction')
ylabel('Elastic Moduli (Gpa)')
hold on;
plot(expr_vf,thenexpr,'g-')
%case 0, when the state of the toggle button is off
   case 0
      cla(handles.plotass);
end
end
if rbcontrol == 2
%normalisation divide results of other models by the rules of mixture
thenrofm = therofm./therofm;
thenhtsai = thehtsai./therofm;
thenmhtsai = themhtsai./therofm;
thenexpr = expr_val./exprofm;

% Switch the state of the toggle button on/off
switch val
%case 1, when the state of the toggle button is on
   case 1
%plot the normalised
%rule of mixture against volume fraction on the first axis
axes(handles.plotass);
plot(vf,thenrofm,'b-')
ylim([-1 3]);
xlabel('volume fraction')
ylabel('Elastic Moduli (Gpa)')
hold on;
plot(vf,thenhtsai,'m-')
plot(vf,thenmhtsai,'k-')
plot(expr_vf,thenexpr,'g-')
%case 0, when the state of the toggle button is off
   case 0
   cla(handles.plotass)
end
end
if rbcontrol == 3
%normalisation divide results of other models by the rules of mixture
thenrofm = therofm./therofm;
thenexpr = expr_val./exprofm;

% Switch the state of the toggle button on/off
switch val
%case 1, when the state of the toggle button is on
   case 1
%plot the normalised rule of mixture against
%volume fraction on the second axis
axes(handles.plotass);
plot(vf,thenrofm,'b-')
xlabel('volume fraction')
ylabel('Elastic Moduli (Gpa)')
ylim([0 15])
hold on;
plot(expr_vf,thenexpr,'g-')
```

```
%case 0, when the state of the toggle button is off
   case 0
   cla(handles.plotass)
end
end
if rbcontrol == 4
%normalisation divide results of other models by the rules of mixture
thenrofm = therofm./therofm;
thenhtsai = thehtsai./therofm;
thenmrofm = themrofm./therofm;
thenexpr = expr_val./exprofm;

% Switch the state of the toggle button on/off
switch val
%case 1, when the state of the toggle button is on
   case 1
%plot the normalised rule of mixture against
%volume fraction on the second axis
axes(handles.plotass);
plot(vf,thenrofm,'b-')
xlabel('volume fraction')
ylabel('Elastic Moduli (Gpa)')
hold on;
plot(vf,thenhtsai,'m-')
plot(vf,thenmrofm,'r-')
plot(expr_vf,thenexpr,'g-')
%case 0, when the state of the toggle button is off
   case 0
   cla(handles.plotass)
end
end


% --- Executes on button press in H_tsai.
function H_tsai_Callback(hObject, eventdata, handles)
%get the name of the fibre and save it to the variable current_mat
%if the fibre box is empty just return
current_mat = get(handles.fibre,'string');
if isempty(current_mat)
   return
end
val = get(hObject,'Value');
% declare global varables required for calculation
% all variables stand for the same thing as in the rule of mixture
% functions
%thehtsai is the variable that returns the results from the
%halpin-tsai.

global  rbcontrol em e2f g12f gm thehtsai;
vf = 0:0.005:0.50;  %a volume fraction vector
eta = 0.5; %based on marcel dekker 1974
etaprime = 1; %based on marcel dekker 1974
if rbcontrol == 2
 %the halpin-tsai function is called to solve for E2 and return the results
 %and save it to the varibale thehtsai.

thehtsai = E2htsai(vf,e2f,em,eta);
```

```
% Switch the state of the toggle button on/off
switch val
%case 1, when the state of the toggle button is on
   case 1
%plot the Halpin-Tsai against volume fraction on the first axis
axes(handles.plotter);
plot(vf,thehtsai,'m-')
xlabel('volume fraction')
ylabel('Elastic Moduli (Gpa)')
grid on
hold on;
%case 0, when the state of the toggle button is off
   case 0
      axes(handles.plotter);
   plot(vf,thehtsai,'w-')
end
end
if rbcontrol == 4
 %the halpin-tsai function is called to solve for G12 and return the
 %results and save it to the varibale thehtsai.
thehtsai = G12htsai(vf,g12f,gm,etaprime);
% Switch the state of the toggle button on/off
switch val
%case 1, when the state of the toggle button is on
   case 1
%plot the Halpin-Tsai against volume fraction on the first axis
axes(handles.plotter);
plot(vf,thehtsai,'k-')
xlabel('volume fraction')
ylabel('Elastic Moduli (Gpa)')
grid on
hold on;
%case 0, when the state of the toggle button is off
   case 0
      axes(handles.plotter);
   plot(vf,thehtsai,'w-')
end
end

% --- Executes on button press of the Normalized Halpin-Tsai.
function nh_tsai_Callback(hObject, eventdata, handles)
%get the name of the fibre and save it to the variable current_mat
%if the fibre box is empty just return
current_mat = get(handles.fibre,'string');
if isempty(current_mat)
   return
end
%get the on/off state of the toggle button and save it to val
val = get(hObject,'Value');

%declare global varables required for calculation
%all variables stand for the same thing as in the previous
%functions
%exphtsai are values extracted from the results of the halpin-tsai
% to match the size of expr_val so they can divide each other.
```

```
global therofm rbcontrol thehtsai expr_vf;
global exphtsai expr_val themrofm themhtsai;
vf = 0:0.005:0.50;  %a volume fraction vector
if rbcontrol == 2
%normalisation divide results of other models by the Halpin Tsai results
thenhtsai = thehtsai./thehtsai;
thenrofm = therofm./thehtsai;
thenmhtsai = themhtsai./thehtsai;
thenexpr = expr_val./exphtsai;

% Switch the state of the toggle button on/off
switch val
%case 1, when the state of the toggle button is on
   case 1
%plot the normalised Halpin Tsai against
%volume fraction on the second axis
axes(handles.plotass);
plot(vf,thenhtsai,'m-')
ylim([-1 3]); %limit y to centre the plot on the screen
xlabel('volume fraction')
ylabel('Elastic Moduli (Gpa)')
hold on;
%plot the other normalised models against
%volume fraction on the second axis
plot(vf,thenrofm,'b-')
plot(vf,thenmhtsai,'k-')
plot(expr_vf,thenexpr,'g-')
%case 0, when the state of the toggle button is off
   case 0
   cla(handles.plotass)
end
end

if rbcontrol == 4
%normalisation divide results of other models by the Halpin Tsai results
thenhtsai = thehtsai./thehtsai;
thenrofm = therofm./thehtsai;
thenmrofm = themrofm./thehtsai;
thenexpr = expr_val./exphtsai;

% Switch the state of the toggle button on/off
switch val
%case 1, when the state of the toggle button is on
   case 1
%plot the normalised halpin-tsai against volume fraction on the first axis
axes(handles.plotass);
plot(vf,thenhtsai,'m-')
xlabel('volume fraction')
ylabel('Elastic Moduli (Gpa)')
hold on;
plot(vf,thenrofm,'b-')
plot(vf,thenmrofm,'r-')
plot(expr_vf,thenexpr,'g-')
%case 0, when the state of the toggle button is off
   case 0
   cla(handles.plotass)
end
```

```
end


% --- Executes on button press of the modified rule of mixtures button
function modr_of_m_Callback(hObject, eventdata, handles)

%get the name of the fibre and save it to the variable current_mat
%if the fibre box is empty just return

current_mat = get(handles.fibre,'string');
if isempty(current_mat)
   return
end
val = get(hObject,'Value');

% declare global varables required for calculation
%all variables stand for the same thing as in the previous functions
%therofm is the vaue from the results of calculation with
%the modified rules of mixture model
%gm is the shear modulus of the matrix

global  rbcontrol g12f gm themrofm;
vf = 0:0.005:0.50;  %a volume fraction vector

if rbcontrol == 4

%the modified rule of mixtures function is called to solve for G12 and
%return the results and save it to the varibale themrofm.

themrofm = G12mrofm(vf,g12f,gm);
% Switch the state of the toggle button on/off
switch val
%case 1, when the state of the toggle button is on
   case 1
%plot the rule of mixture against volume fraction on the first axis
axes(handles.plotter);
plot(vf,themrofm,'r-')
xlabel('volume fraction')
ylabel('Elastic Moduli (Gpa)')
grid on
hold on;
%case 0, when the state of the toggle button is off
   case 0
      axes(handles.plotter);
      plot(vf,themrofm,'w-')
end
end

%Executes on press of the normalized modified rule of mixtures button
function nmodr_of_m_Callback(hObject, eventdata, handles)

%get the name of the fibre and save it to the variable current_mat
%if the fibre box is empty just return

current_mat = get(handles.fibre,'string');
```

```
if isempty(current_mat)
   return
end

%get the on/off state of the toggle button and save it to val
val = get(hObject,'Value');

%declare global variables required for calculation
%all variables stand for the same thing as in the previous functions
%expmrofm are values extracted from the results of the modified rule of
%mixtures to match the size of expr_val so they can divide each other.

global therofm rbcontrol thehtsai expr_val expr_vf expmrofm themrofm;
vf = 0:0.005:0.50;  %a volume fraction vector
if rbcontrol == 4

%normalisation divide results of other models by the
%modified rule of mixture results

thenhtsai = thehtsai./themrofm;
thenrofm = therofm./themrofm;
thenmrofm = themrofm./themrofm;
thenexpr = expr_val./expmrofm;

% Switch the state of the toggle button on/off
switch val
%case 1, when the state of the toggle button is on
   case 1

%plot the normalised modified rule of mixture
%against volume fraction on the first axis

axes(handles.plotass);
plot(vf,thenmrofm,'r-')
xlabel('volume fraction')
ylabel('Elastic Moduli (Gpa)')
hold on;
plot(vf,thenrofm,'b-')
plot(vf,thenhtsai,'m-')
plot(expr_vf,thenexpr,'g-')
%case 0, when the state of the toggle button is off
   case 0
   cla(handles.plotass)
end
end

% --- Executes on button press in modhtsai.
function modhtsai_Callback(hObject, eventdata, handles)

%get the name of the fibre and save it to the variable current_mat
%if the fibre box is empty just return

current_mat = get(handles.fibre,'string');
if isempty(current_mat)
   return
end
```

```
val = get(hObject,'Value');

% declare global varables required for calculation
%all variables stand for the same thing as in the previous functions
%themhtsai is the vaue from the results of calculation with
%the modified Halpin-Tsai model
%num is the poisson ratio of the matrix
%nu12f is the poisson ratio of the fibre in the 12 direction

global  rbcontrol e1f e2f em nu12f num themhtsai;
vf = 0:0.005:0.50;  %a volume fraction vector

% In most cases nu12f is equal to nu21f

nu21f = nu12f;
if rbcontrol == 2

 %the modified Halpin-Tsai function is called to solve for E2 and
%return the results and save it to the varibale themhtsai.

themhtsai = E2mhtsai(vf,e2f,em,nu12f,nu21f,num,e1f);
% Switch the state of the toggle button on/off
switch val
%case 1, when the state of the toggle button is on
   case 1

%plot the modified rule of mixtures against
%volume fraction on the first axis

axes(handles.plotter);
plot(vf,themhtsai,'k-')
xlabel('volume fraction')
ylabel('Elastic Moduli (Gpa)')
grid on
hold on;
%case 0, when the state of the toggle button is off
   case 0
      axes(handles.plotter);
      plot(vf,themhtsai,'w-')
end
end

% --- Executes on button press in nmodh_tsai.
function nmodh_tsai_Callback(hObject, eventdata, handles)

%get the name of the fibre and save it to the variable current_mat
%if the fibre box is empty just return

current_mat = get(handles.fibre,'string');
if isempty(current_mat)
   return
end
%get the on/off state of the toggle button and save it to val
val = get(hObject,'Value');
```

```
% declare global variables required for calculation
%all variables stand for the same thing as in the previous functions
%expmhtsai are values extracted from the results of the modified Halpin
%Tsai to match the size of expr_val so they can divide each other.

global therofm rbcontrol thehtsai themhtsai expr_vf expr_val expmhtsai;
vf = 0:0.005:0.50;  %a volume fraction vector
if rbcontrol == 2

 %normalisation divide results of other models by the
%modified Halpin Tsai results

thenmhtsai = themhtsai./themhtsai;
thenhtsai = thehtsai./themhtsai;
thenrofm = therofm./themhtsai;
thenexpr = expr_val./expmhtsai;
% Switch the state of the toggle button on/off
switch val
%case 1, when the state of the toggle button is on
   case 1
%plot the normalized Halpin-Tsai against volume fraction on the second axis
axes(handles.plotass);
plot(vf,thenmhtsai,'k-')
ylim([-1 3]);
xlabel('volume fraction')
ylabel('Elastic Moduli (Gpa)')
hold on;

%Plot the other normalised results against volume fraction on the axis
plot(vf,thenrofm,'b-')
plot(vf,thenhtsai,'m-')
plot(expr_vf,thenexpr,'g-')
%case 0, when the state of the toggle button is off
   case 0
   cla(handles.plotass)
end
end

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


% --- Executes on button press in Expr.
function Expr_Callback(hObject, eventdata, handles)

%get the state of the toggle button and save it to the variable val.

val = get(hObject,'Value');

%declare global variables required for calculation
%all variables stand for the same thing as in the previous functions
%expr_vf is the volume fraction associated with the experimental values as
%saved in the database
%expr_val are the experimental values of the mechanical property
%associated with the experimental values as saved in the database

global expr_vf rbcontrol expr_val;
```

```
global therofm thehtsai themrofm themhtsai;
global exprofm exphtsai expmrofm expmhtsai;

%get the name of the composite and save it to the variable
%current_composite, the name is saved in a static text box on th GUI with
%the tag composite so you get it through handles.composite

current_composite = get(handles.composite,'string');
%if the current composite is empty return
if isempty(current_composite)
   return
end

%save the experimental data from the database to a variable expr, no
%programming significance, just a shorter name.

expr = handles.Experimental;
for i = 1:size(expr,1)

   %compare the name of the composite with the database composites
   %strcmpi for non-case sensitive
   %if the composite exist in the database check the value of rbcontrol
   %and determine what mechanical property is required and get the values
   %and save them to expr_val and save the corresponding volume fraction
   %to expr_vf
   %use the cell2mat function to convert the database data from cell format
 %to matrix format to be used for calculations.

if strcmpi(expr(i,1),current_composite)
      if rbcontrol == 1 %check what mechanical property
expr_vf(1) = cell2mat(expr(i,3));    expr_val(1) = cell2mat(expr(i,4));
expr_vf(2) = cell2mat(expr(i,5));    expr_val(2) = cell2mat(expr(i,6));
expr_vf(3) = cell2mat(expr(i,7));    expr_val(3) = cell2mat(expr(i,8));
expr_vf(4) = cell2mat(expr(i,9));    expr_val(4) = cell2mat(expr(i,10));
expr_vf(5) = cell2mat(expr(i,11));   expr_val(5) = cell2mat(expr(i,12));
expr_vf(6) = cell2mat(expr(i,13));   expr_val(6) = cell2mat(expr(i,14));
expr_vf(7) = cell2mat(expr(i,15));   expr_val(7) = cell2mat(expr(i,16));
expr_vf(8) = cell2mat(expr(i,17));   expr_val(8) = cell2mat(expr(i,18));
expr_vf(9) = cell2mat(expr(i,19));   expr_val(9) = cell2mat(expr(i,20));
expr_vf(10) = cell2mat(expr(i,21));  expr_val(10) = cell2mat(expr(i,22));
expr_vf(11) = cell2mat(expr(i,23));  expr_val(11) = cell2mat(expr(i,24));

%save the comment and reference associated with that set of
%experimental data to variables comments and references respectively and
%set the handles of two static textboxes on the GUI to show the data.

   comments = expr(i,25);      set(handles.comment,'string',comments)
   references = expr(i,26);    set(handles.reference,'string',references)
        switch val
        %case 1, when the state of the toggle button is on
         case 1
 %plot the experimental data against volume fraction on the first axis
            axes(handles.plotter);
            plot(expr_vf,expr_val,'gv')
            xlabel('volume fraction')
            ylabel('Elastic Moduli (Gpa)')
```

```
            grid on
            hold on;
        %case 0, when the state of the toggle button is off
        case 0
            axes(handles.plotter);
            plot(expr_vf,expr_val,'wv')
             set(handles.comment,'string','')
            set(handles.reference,'string','')
        end

%this loop extracts 11 values of rule of mixtures @ matching volume
%fractions to the experimental value from therofm and saves it to exprofm
%for use in normilisation functions

for j = 1:11
    exprofm(j) = therofm(j+(9*(j-1)));
end
      end

       if rbcontrol == 2
expr_vf(1) = cell2mat(expr(i+1,3));    expr_val(1) = cell2mat(expr(i+1,4));
expr_vf(2) = cell2mat(expr(i+1,5));    expr_val(2) = cell2mat(expr(i+1,6));
expr_vf(3) = cell2mat(expr(i+1,7));    expr_val(3) = cell2mat(expr(i+1,8));
expr_vf(4) = cell2mat(expr(i+1,9));    expr_val(4) = cell2mat(expr(i+1,10));
expr_vf(5) = cell2mat(expr(i+1,11));   expr_val(5) = cell2mat(expr(i+1,12));
expr_vf(6) = cell2mat(expr(i+1,13));   expr_val(6) = cell2mat(expr(i+1,14));
expr_vf(7) = cell2mat(expr(i+1,15));   expr_val(7) = cell2mat(expr(i+1,16));
expr_vf(8) = cell2mat(expr(i+1,17));   expr_val(8) = cell2mat(expr(i+1,18));
expr_vf(9) = cell2mat(expr(i+1,19));   expr_val(9) = cell2mat(expr(i+1,20));
expr_vf(10) = cell2mat(expr(i+1,21));  expr_val(10) = cell2mat(expr(i+1,22));
expr_vf(11) = cell2mat(expr(i+1,23));  expr_val(11) = cell2mat(expr(i+1,24));
   comments = expr(i+1,25);       set(handles.comment,'string',comments)
   references = expr(i+1,26);    set(handles.reference,'string',references)
        switch val
        %case 1, when the state of the toggle button is on
         case 1
   %plot the experimental data against volume fraction on the first axis
            axes(handles.plotter);
            plot(expr_vf,expr_val,'gv')
            xlabel('volume fraction')
            ylabel('Elastic Moduli (Gpa)')
            grid on
            hold on;
        %case 0, when the state of the toggle button is off
        case 0
            axes(handles.plotter);
            plot(expr_vf,expr_val,'wv')
             set(handles.comment,'string','')
            set(handles.reference,'string','')
        end

%this loop extracts 11 values of rule of mixtures @ matching volume
%fractions to the experimental value from therofm and saves it to exprofm
%for use in normilisation functions. the exphtsai and expmhtsai are
%similar they extract 11 values from halpin tsai and modified halpin tsai
%respectively.
```

```
for j = 1:11
   exprofm(j) = therofm(j+(9*(j-1)));
end
for j = 1:11
   exphtsai(j) = thehtsai(j+(9*(j-1)));
end
for j = 1:11
   expmhtsai(j) = themhtsai(j+(9*(j-1)));
end
      end

      if rbcontrol == 3
expr_vf(1) = cell2mat(expr(i+2,3));    expr_val(1) = cell2mat(expr(i+2,4));
expr_vf(2) = cell2mat(expr(i+2,5));    expr_val(2) = cell2mat(expr(i+2,6));
expr_vf(3) = cell2mat(expr(i+2,7));    expr_val(3) = cell2mat(expr(i+2,8));
expr_vf(4) = cell2mat(expr(i+2,9));    expr_val(4) = cell2mat(expr(i+2,10));
expr_vf(5) = cell2mat(expr(i+2,11));   expr_val(5) = cell2mat(expr(i+2,12));
expr_vf(6) = cell2mat(expr(i+2,13));   expr_val(6) = cell2mat(expr(i+2,14));
expr_vf(7) = cell2mat(expr(i+2,15));   expr_val(7) = cell2mat(expr(i+2,16));
expr_vf(8) = cell2mat(expr(i+2,17));   expr_val(8) = cell2mat(expr(i+2,18));
expr_vf(9) = cell2mat(expr(i+2,19));   expr_val(9) = cell2mat(expr(i+2,20));
expr_vf(10) = cell2mat(expr(i+2,21));  expr_val(10) = cell2mat(expr(i+2,22));
expr_vf(11) = cell2mat(expr(i+2,23));  expr_val(11) = cell2mat(expr(i+2,24));
  comments = expr(i+2,25);      set(handles.comment,'string',comments)
  references = expr(i+2,26);    set(handles.reference,'string',references)
       switch val
       %case 1, when the state of the toggle button is on
        case 1
     %plot the rule of mixture against volume fraction on the first axis
          axes(handles.plotter);
          plot(expr_vf,expr_val,'gv')
          xlabel('volume fraction')
          ylabel('Elastic Moduli (Gpa)')
          grid on
          hold on;
       %case 0, when the state of the toggle button is off
       case 0
          axes(handles.plotter);
          plot(expr_vf,expr_val,'wv')
           set(handles.comment,'string','')
          set(handles.reference,'string','')
       end
for j = 1:11
   exprofm(j) = therofm(j+(9*(j-1)));
end
      end

      if rbcontrol == 4
expr_vf(1) = cell2mat(expr(i+3,3));    expr_val(1) = cell2mat(expr(i+3,4));
expr_vf(2) = cell2mat(expr(i+3,5));    expr_val(2) = cell2mat(expr(i+3,6));
expr_vf(3) = cell2mat(expr(i+3,7));    expr_val(3) = cell2mat(expr(i+3,8));
expr_vf(4) = cell2mat(expr(i+3,9));    expr_val(4) = cell2mat(expr(i+3,10));
expr_vf(5) = cell2mat(expr(i+3,11));   expr_val(5) = cell2mat(expr(i+3,12));
expr_vf(6) = cell2mat(expr(i+3,13));   expr_val(6) = cell2mat(expr(i+3,14));
expr_vf(7) = cell2mat(expr(i+3,15));   expr_val(7) = cell2mat(expr(i+3,16));
expr_vf(8) = cell2mat(expr(i+3,17));   expr_val(8) = cell2mat(expr(i+3,18));
expr_vf(9) = cell2mat(expr(i+3,19));   expr_val(9) = cell2mat(expr(i+3,20));
```

```
expr_vf(10) = cell2mat(expr(i+3,21));   expr_val(10) = cell2mat(expr(i+3,22));
expr_vf(11) = cell2mat(expr(i+3,23));   expr_val(11) = cell2mat(expr(i+3,24));
comments = expr(i+3,25);       set(handles.comment,'string',comments)
references = expr(i+3,26);     set(handles.reference,'string',references)
        switch val
        %case 1, when the state of the toggle button is on
         case 1
      %plot the rule of mixture against volume fraction on the first axis
           axes(handles.plotter);
           plot(expr_vf,expr_val,'gv')
           xlabel('volume fraction')
           ylabel('Elastic Moduli (Gpa)')
           grid on
           hold on;
        %case 0, when the state of the toggle button is off
        case 0
           axes(handles.plotter);
           plot(expr_vf,expr_val,'wv')
            set(handles.comment,'string','')
           set(handles.reference,'string','')
        end

for j = 1:11
   exprofm(j) = therofm(j+(9*(j-1)));
end
for j = 1:11
   exphtsai(j) = thehtsai(j+(9*(j-1)));
end

%this loop extracts 11 values of modified rule of mixtures @ matching
%volume fractions to the experimental value from themrofm and saves
% it to expmrofm for use in normilisation functions.

for j = 1:11
   expmrofm(j) = themrofm(j+(9*(j-1)));
end
      end



end
end

if isempty(expr)
%if the experimental data for the material cannot be found in the database
%display a message box with the message below
msgbox('The experimental data for this composite is not on the database. To add it to the database click on Data
management on the menu bar', ...
   'Unknown Material', 'help');
return
end



% --- Executes on button press in nexpr.
function nexpr_Callback(hObject, eventdata, handles)

%get the name of the fibre and save it to the variable current_mat
```

```
%if the fibre box is empty just return

current_mat = get(handles.fibre,'string');
if isempty(current_mat)
   return
end
%get the on/off state of the toggle button and save it to val
val = get(hObject,'Value');
% declare global varables required for calculation
%all variables stand for the same thing as in the previous functions

global rbcontrol expr_val expr_vf exprofm exphtsai expmrofm expmhtsai;

if rbcontrol == 1
thenrofm = exprofm./expr_val;
thenexpr = expr_val./expr_val;
switch val
%case 1, when the state of the toggle button is on
   case 1
%plot the rule of mixture against volume fraction on the first axis
axes(handles.plotass);
plot(expr_vf,thenexpr,'g-')
ylim([-1 3]);
xlabel('volume fraction')
ylabel('Elastic Moduli (Gpa)')
hold on;
plot(expr_vf,thenrofm,'b-')
%case 0, when the state of the toggle button is off
   case 0
      cla(handles.plotass);
end
end

if rbcontrol == 2
thenhtsai = exphtsai./expr_val;
thenmhtsai = expmhtsai./expr_val;
thenrofm = exprofm./expr_val;
thenexpr = expr_val./expr_val;
switch val
%case 1, when the state of the toggle button is on
   case 1
%plot the rule of mixture against volume fraction on the first axis
axes(handles.plotass);
plot(expr_vf,thenexpr,'g-')
ylim([-1 3]);
xlabel('volume fraction')
ylabel('Elastic Moduli (Gpa)')
hold on;
plot(expr_vf,thenrofm,'b-')
plot(expr_vf,thenhtsai,'m-')
plot(expr_vf,thenmhtsai,'k-')
%case 0, when the state of the toggle button is off
   case 0
      cla(handles.plotass);
end
end
```

```
if rbcontrol == 3
thenrofm = exprofm./expr_val;
thenexpr = expr_val./expr_val;
switch val
%case 1, when the state of the toggle button is on
   case 1
%plot the rule of mixture against volume fraction on the first axis
axes(handles.plotass);
plot(expr_vf,thenexpr,'g-')
ylim([-1 3]);
xlabel('volume fraction')
ylabel('Elastic Moduli (Gpa)')
hold on;
plot(expr_vf,thenrofm,'b-')
%case 0, when the state of the toggle button is off
   case 0
      cla(handles.plotass);
end
end

if rbcontrol == 4
thenmrofm = expmrofm./expr_val;
thenhtsai = exphtsai./expr_val;
thenrofm = exprofm./expr_val;
thenexpr = expr_val./expr_val;
switch val
%case 1, when the state of the toggle button is on
   case 1
%plot the rule of mixture against volume fraction on the first axis
axes(handles.plotass);
plot(expr_vf,thenexpr,'g-')
ylim([-1 3]);
xlabel('volume fraction')
ylabel('Elastic Moduli (Gpa)')
hold on;
plot(expr_vf,thenrofm,'b-')
plot(expr_vf,thenhtsai,'m-')
plot(expr_vf,thenmrofm,'r-')
%case 0, when the state of the toggle button is off
   case 0
      cla(handles.plotass);
end
end


function fibre_Callback(hObject, eventdata, handles)
global e1f e2f g12f g23f nu12f nu23f;

%Get the fibre name from the fibre editable textbox and save to current_mat
%clean up all spaces before or after the name with strtrim

current_mat = get(handles.fibre,'string');
current_mat = strtrim(current_mat);
if isempty(current_mat)
   return
end
% Get the current list of fibres from the handles structure and save it to
```

```
% the variable Fibres

 Fibres = handles.Fibres;

 %compare the name of the fibre with the database fibres
 %if the fibre exist in the database, get the values of its mechanical
 %properties and save them to the matching variable name.
%use the cell2mat function to convert the database data from cell format
 %to matrix format to be used for calculations.

for i = 1:length(Fibres)
   %strcmpi for non-case sensitive
   if strcmpi(Fibres(i,1),current_mat)
      set(handles.fibre,'string',Fibres(i,1))
      fibre_den = cell2mat(Fibres(i,2));
      fibre_dia = cell2mat(Fibres(i,3));
      e1f = cell2mat(Fibres(i,4));
      e2f = cell2mat(Fibres(i,5));
      g12f = cell2mat(Fibres(i,6));
      g23f = cell2mat(Fibres(i,7));
      nu12f = cell2mat(Fibres(i,8));
       nu23f = cell2mat(Fibres(i,9));
      handles.Index = i;
      guidata(hObject,handles)
      return
   end
end
%if the material does not exist in the database
%a message box with the message below will pop up
msgbox('This material is not on the database. To add it to the database click on Data  management on the menu
bar', ...
   'Unknown Material', 'help');

% --- Executes during object creation, after setting all properties.
function fibre_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end



% --- Executes on button press in comp_ok_btn.
function comp_ok_btn_Callback(hObject, eventdata, handles)

%Get the composite name from the composite static textbox and save to
%current_comp. This value is initially an empty string.
%Get the fibre name from the fibre editable textbox and save to
%current_fibre
%Get the matrix name from the matrix editable textbox and save to
%current_matrix
%create a composite name by catenating the name of the fibre and matrix
%with a dash in the middle
%set the composite static textbox to display the name of the composite

current_comp = get(handles.composite,'string');
current_fibre = get(handles.fibre,'string');
```

```
current_matrix = get(handles.matrix,'string');
current_comp = strcat(current_fibre,'-',current_matrix);
 set(handles.composite,'string',current_comp)



function matrix_Callback(hObject, eventdata, handles)
global em gm num;

%Get the matrix name from the matrix editable textbox and save
%to current_mat
%clean up all spaces before or after the name with strtrim

current_mat = get(handles.matrix,'string');
current_mat = strtrim(current_mat);
if isempty(current_mat)
   return
end

% Get the current list of matrices from the handles structure and save
%it to the variable Matrices

Matrices = handles.Matrices;

 %compare the name of the matrix with the database matrices
 %if the matrix exist in the database, get the values of its mechanical
 %properties and save them to the matching variable name.
 %use the cell2mat function to convert the database data from cell format
 %to matrix format to be used for calculations.

for i = 1:length(Matrices)
   %strcmpi for non-case sensitive
   if strcmpi(Matrices(i,1),current_mat)
      set(handles.matrix,'string',Matrices(i,1))
      mat_den = cell2mat(Matrices(i,2));
      em = cell2mat(Matrices(i,3));
      gm = cell2mat(Matrices(i,4));
      num = cell2mat(Matrices(i,5));
      handles.Index = i;
      guidata(hObject,handles)
      return
   end
end
%if the material does not exist in the database
%a message box with the message below will pop up
msgbox('This material is not on the database. To add it to the database click on Data  management on the menu
bar', ...
   'Unknown Material', 'help');



% --- Executes during object creation, after setting all properties.
function matrix_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
```

```
end


% code to execute when the file menu is clicked in this case nothing
function file_menu_Callback(hObject, eventdata, handles)

% code to execute when the close sub menu under the file menu is clicked
%closes the program
function close_menu_Callback(hObject, eventdata, handles)
close(handles.Compositor);

% code to execute when the help menu is clicked
function help_menu_Callback(hObject, eventdata, handles)
%opens the Help file in a browser window
HelpPath = which('compositor_help.html');
web(HelpPath);



% --- Executes on button press in schap.
function schap_Callback(hObject, eventdata, handles)
%Reserved for future Use


% --- Executes on button press in nschap.
function nschap_Callback(hObject, eventdata, handles)
%Reserved for future Use

% --- Executes during object creation, after setting all properties.
function fibre_data_table_CreateFcn(hObject, eventdata, handles)

% --- Executes when entered data in editable cell(s) in fibre_data_table.
function fibre_data_table_CellEditCallback(hObject, eventdata, handles)

%gets the data of the fibre data table through the handle structure and
%saves it to the variable fibre_data

fibre_data = get(handles.fibre_data_table, 'Data');

%saves the fibre_data variable to the handle structure for global usage

handles.Fibres = fibre_data;
% Update handles structure
guidata(hObject, handles);


% --- Executes when selected cell(s) is changed in fibre_data_table.
function fibre_data_table_CellSelectionCallback(hObject, eventdata, handles)



% --- Executes when entered data in editable cell(s) in matrix_data_table.
function matrix_data_table_CellEditCallback(hObject, eventdata, handles)
```

```
%gets the data of the matrix data table through the handle structure and
%saves it to the variable matrix_data

matrix_data = get(handles.matrix_data_table, 'Data');

%saves the matrix_data variable to the handle structure for global usage

 handles.Matrices = matrix_data;
% Update handles structure
guidata(hObject, handles);


%Executes when entered data in editable cell(s) in experimental_data_table.
function experimental_data_table_CellEditCallback(hObject, eventdata, handles)

%gets the data of the experimental data table through the handle
%structure and saves it to the variable experimental_data

experimental_data = get(handles.experimental_data_table, 'Data');

%saves the experimental_data variable to the
%handle structure for global usage

handles.Experimental = experimental_data;
% Update handles structure
guidata(hObject, handles);

% --- Executes on button press in fibre_new_button.
function fibre_new_button_Callback(hObject, eventdata, handles)

%retrieves the fibre_data from the handles structure through
%handles.Fibres


fibre_data = handles.Fibres;

%creates a new cell array with one cell at the end of every column of the
%table

fibre_data(end+1,:)= cell(1);

%edits the new row and sets the first column to be a character array

fibre_data(end,1) = {'material'};

%saves the data in the new row to the handles structure and displays it on
%screen with the set function

set(handles.fibre_data_table,'Data',fibre_data)
% Update handles structure
guidata(hObject, handles);

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

```
%Use matlab's built in xlswrite to write to the excel database file and
%permanently save any changes or additions made in the program
%For error check save the status to status and any error messages to
%message

[status, message] = xlswrite('materials_data.xls', handles.Fibres,'Fibres');

%if the save was not successful output the error message

if status == 0
msgbox(message.message);

%else if save was successful display a message box with the message below

else
   msgbox('Data was saved sucessfully to the Database','successful');
end

% --- Executes on button press in matrix_new_button.
function matrix_new_button_Callback(hObject, eventdata, handles)

%same as fibre new button but for the matrix material

matrix_data = handles.Matrices;
matrix_data(end+1,:)= cell(1);
matrix_data(end,1) = {'material'};
set(handles.matrix_data_table,'Data',matrix_data)
% Update handles structure
guidata(hObject, handles);

% --- Executes on button press in matrix_save_button.
function matrix_save_button_Callback(hObject, eventdata, handles)

%same as fibre save button but for the matrix material

[status, message] = xlswrite('materials_data.xls',handles.Matrices, 'Matrices');
if status == 0
msgbox(message.message);
else
   msgbox('Data was saved sucessfully to the Database','successful');
end

% --- Executes on button press in experimental_new_button.
function experimental_new_button_Callback(hObject, eventdata, handles)

%similar to the fibre new button but for the experimental data
%In this case four new rows are created for the different mechanical
%properties associated with the experimental data. i.e. Longitudinal
%elastic modulus, Transverse Elastic modulus shear modulus and
%poisson's ratio
%the first two and last two columns are converted from cell to string data
%by initialisation.

experimental_data = handles.Experimental;
```

```
experimental_data(end+1,:)= cell(1);
experimental_data(end,1)= {'material'};
experimental_data(end,2)= {'property'};
experimental_data(end,end - 1)= {'comment'};
experimental_data(end,end)= {'references'};
set(handles.experimental_data_table,'Data',experimental_data)

experimental_data(end+1,:)= cell(1);
experimental_data(end,1)= {'_'};
experimental_data(end,2)= {'property'};
experimental_data(end,end - 1)= {'comment'};
experimental_data(end,end)= {'references'};
set(handles.experimental_data_table,'Data',experimental_data)

experimental_data(end+1,:)= cell(1);
experimental_data(end,1)= {'_'};
experimental_data(end,2)= {'property'};
experimental_data(end,end - 1)= {'comment'};
experimental_data(end,end)= {'references'};
set(handles.experimental_data_table,'Data',experimental_data)

experimental_data(end+1,:)= cell(1);
experimental_data(end,1)= {'_'};
experimental_data(end,2)= {'property'};
experimental_data(end,end - 1)= {'comment'};
experimental_data(end,end)= {'references'};
set(handles.experimental_data_table,'Data',experimental_data)
% Update handles structure
guidata(hObject, handles);

% --- Executes on button press in experimental_save_button.
function experimental_save_button_Callback(hObject, eventdata, handles)

%same as fibre new button but for the Experimental

[status, message] = xlswrite('materials_data.xls', handles.Experimental,'Experimental');
if status == 0
msgbox(message.message);
else
  msgbox('Data was saved sucessfully to the Database','successful');
end
```