MEB CALCULATOR

Group 59

December 7, 2015

Contributors:

Ume-Ugwa Chukwubuikem

Luc-Andre Williams

Kyle Robinson

### Abstract

This software is design to compute process variables of a gas using the ideal and virial truncated equation of states, and also to do linear interpolation of data. The software requires that the user provides process specification variables such as temperature, pressure, mass flow rate (open system) etc and computes other process variables based on these specifications. On completion of the computation, the software outputs two tables containing the values of the computed process variables, and a plot of temperature versus the molar volume. One table contains variable values obtained using the ideal equation of state and the other contains the values obtained using the virial truncated equation of states.

### Introduction

In dealing with material and energy balance problems, the process of setting up the problem, and computing it can prove to be a very time consuming and arduous process with a high probability of error. Even when using a computer algebra system like Matlab the process of manually inputting each parameter and equation to be used can be very time consuming as well. To solve this issue and save time when dealing with these types of problems, programs and function can be written which can help with the computation. For this project the computation of process variables of a process gas was address. This was done by computing molar specific volume, velocity, flow rate, density, and energy loss using process specifications such as temperature, mass flow rate, pressure, pipe length, pipe diameter, and the friction factor. The computations were accomplished by leveraging the ideal gas and Virial EOS. A GUI was developed and the necessary input is obvious which makes it very user friendly and more preferable. The scope of the project was limited to 14 different gases. These 14 gases were chosen due to being commonly used gases and their Pitzer acentric factor has already been determined by professionals. New gases can always be incorporated as the software gets bigger and more sophisticated.

### THEORY

The idea behind the creation of this software is to make doing material and energy balance calculations less time consuming and also give the user an idea of how properties of a gas moves towards ideality when the conditions for ideality are been approach.

This software has two parts to it, Calculation of a process gas variable using ideal and virial truncated equation of states and linear interpolation of data.

Calculation of a process gas variable using EOS:

To do this calculation, the software requires that the user enter the Compound name, the temperatures, pressure, length, diameter, friction factor and mass flow rates. All these data has to be in SI units.

The software currently support only fourteen gaseous element/compounds (listed in the appendix)

When the user provides the software with the requested values and the compute button pressed, the software calculates the following process variables using the Ideal and Virial EOS.

* Specific molar volume
* Volumetric flow rate
* Velocity
* Density
* Energy density

The name of the compound is very crucial because the software uses it to retrieve the right critical temperature and pressure, molecular weight and Pitzer acentric factor of the compound. These data are used to estimate virial two body interaction constant, which if not correctly estimated would throw the calculation off. The software does this tedious estimation under the hood without the knowledge of the user.

There are two forms of virial EOS but the software utilizes the most versatile form of the virial equation of state to do the calculations. Form one of the equation which is a better estimate of the molar specific volume for any type of compound is utilized by the software. It involves solving a quadratic and the software handles the quadratic very efficiently. Form one of virial EOS gives a more precise estimate for polar compounds. Form two of the equation is a good estimate for nonpolar compounds and it is not utilized by the software. Form two of the equation was develop for convenience so it is unnecessary to incorporate it into the software since form one gets the job done.

The ideal gas equation of states is also utilized by the software to calculate molar specific volume, in this case it assumes the gas is ideal. The ideal EOS was incorporated in the software for comparison. The ideal EOS gives a better estimate of the molar specific volume when the gas is at high temperature and low pressures (3-5atm). Below are the equations utilized by the software.

The Ideal Gas equation of state:

This equation is a good estimate of the gas process variable when ideal conditions are met such as high temperature and low pressure.

The Virial Truncated Equation of state:

Form 1: Good estimate for both polar and nonpolar gases

Form 2(not utilized): Good estimate for nonpolar gases

The estimate for the energy density is done using this formula

µloss

The length and diameter are property of the pipe used to transport the gas and necessary to estimate the energy losses and f is the friction factor.

On completion of the computation, the software outputs a table of data and plots the specific molar volume versus temperature. The plot is a great visual because it shows how the molar volumes calculated using the two equations approach each other as the temperature increases

Linear Interpolation:

As chemical engineers we will be working with a lot of tables and sometimes the data point of interest is not on the table so we have to interpolate the available data to get this data point. This part of the software reduces the steps that you have to take to accomplish this and also does it a little bit faster.

To use this feature of the software, the user has to provide five data points and the software computes the corresponding value at the point of interest.

### Objectives

This program takes data from an Excel spreadsheet, and uses the data in order to solve Material and Energy Balance problems. When solving Material and Energy Balance problems, there are two common equations of state that can be applied. The main objective of this program is to extract the correct data (dependent upon the compound) and input the data into an equation of state. After using the equations of state to solve for different process variables (volumetric flow rate, , , , ), the program returns 2 tables displaying the data using the Ideal Gas Equation and the Virial Equation. Both results are then plotted against each other to show how the returned values vary.

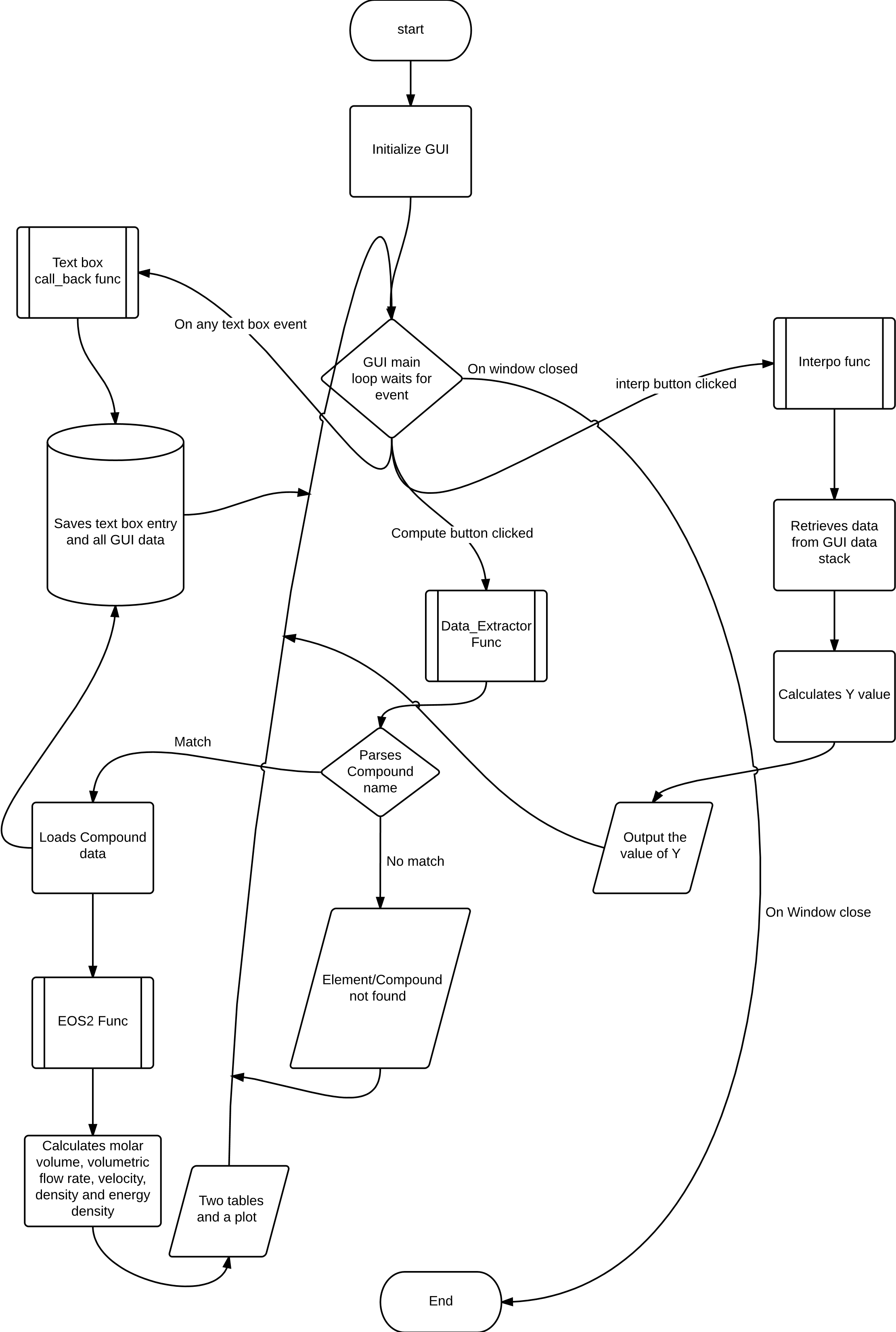
**Main Objectives:**

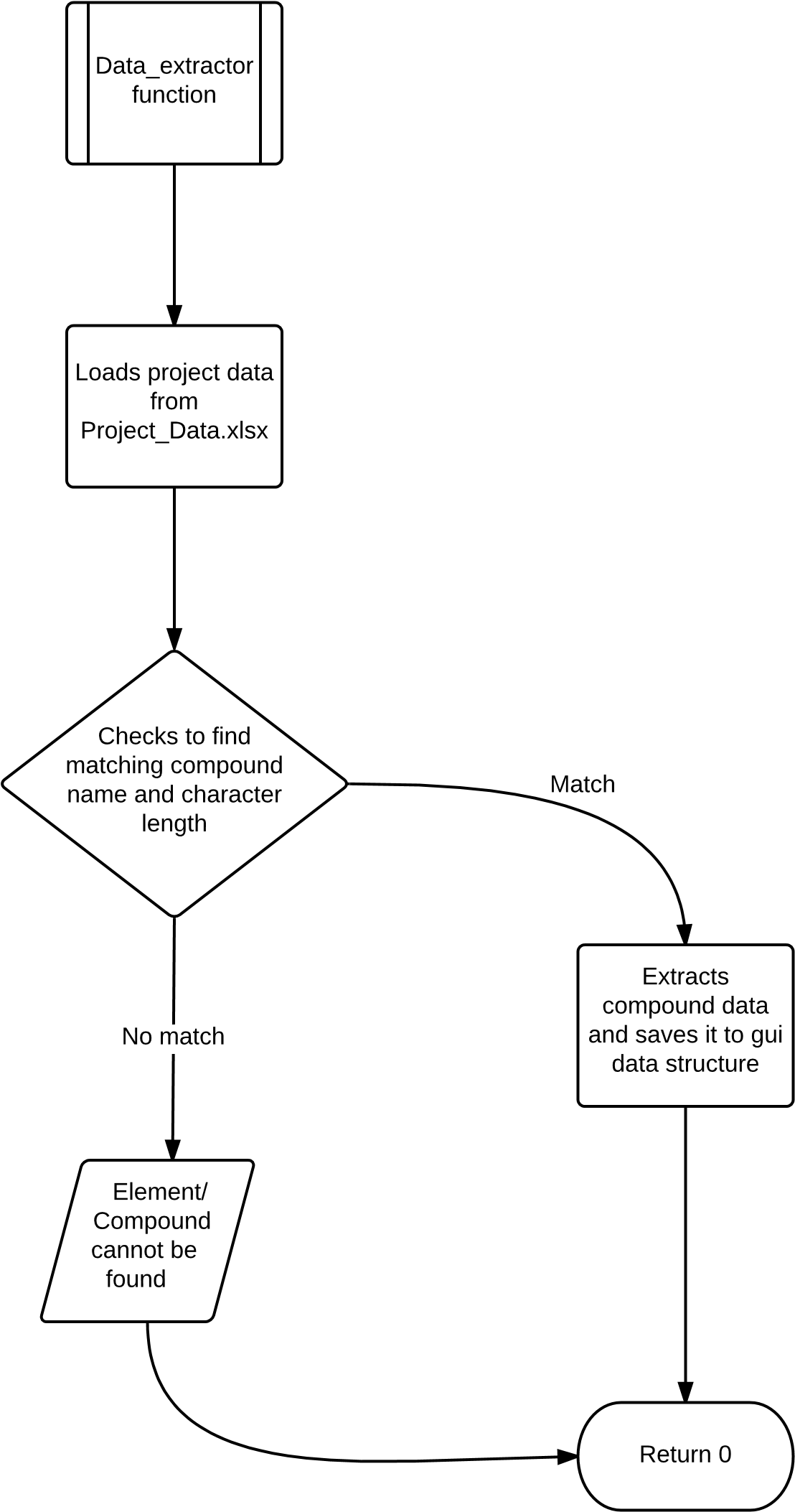
1. Make Material and Energy Balance problems easier to solve
2. Quickly identify variables
3. Shorten the amount of time used to solve problems
4. Verify answers to Material and Energy Balance problems

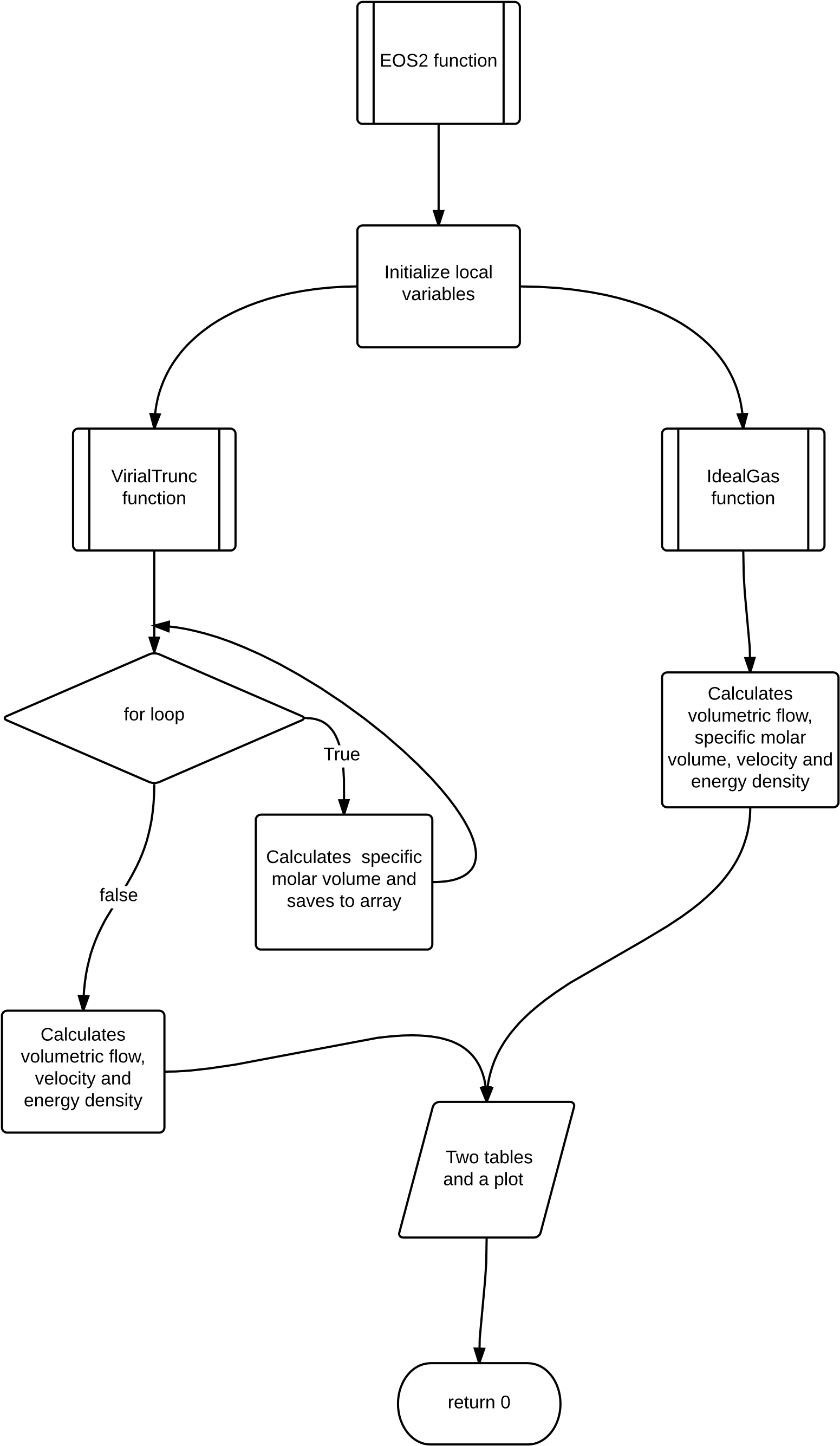
### DESIGN

The software has two parts to it, the process variable calculator and the interpolator. To perform any of these operations two buttons and data entry text boxes has been provided. The Graphical user interface is very detailed so the user basically know where to input data.

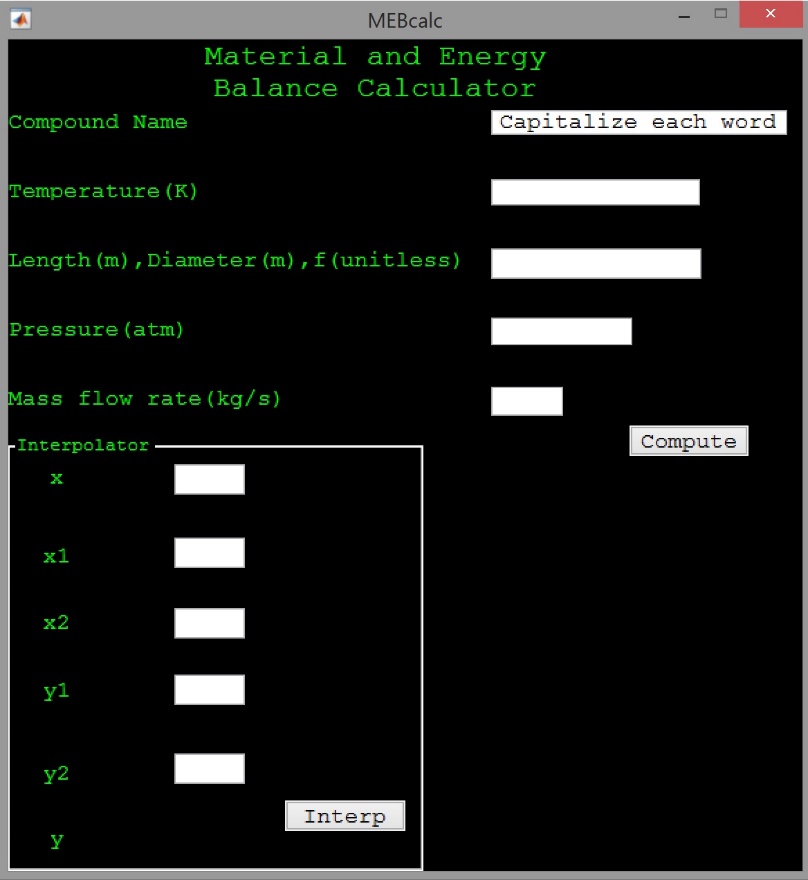
The compute button is used when calculations on process variables is to be performed provided that the process specification values has been entered. While the interp button is used when linear interpolation of data is of interest and data is provided. Below is a flow chart of core functionality of the software and flowcharts for some of the functions used by the software.





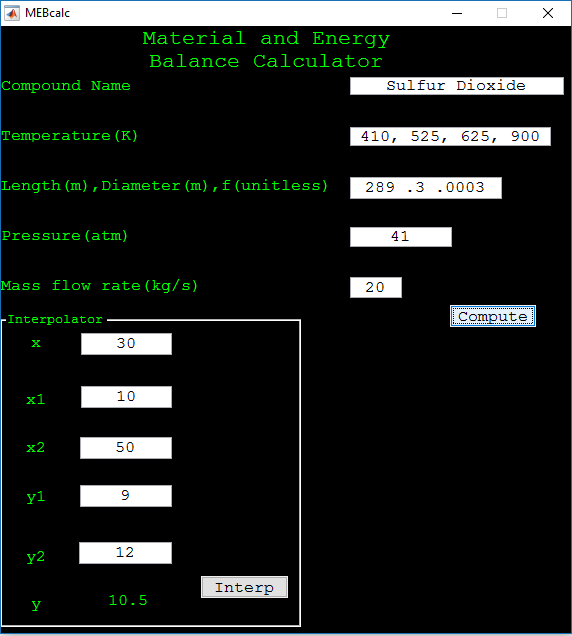


### Screen Shots



**Figure 1.1**

The interface contains boxes for input data needed to solve Material and Energy Balance problems. These inputs are: the compound name, the temperature (in degrees Kelvin), length and diameter of the pipe that the compound is traveling through, unit-less friction factor, the pressure of the fluid in atmospheres, and the mass flow rate in kg/s. The interface also contains a built in interpolator so the user can easily obtain values for data to be entered into the interface.



**Figure 1.2**

This is an example of data entered into the calculator. This would work for a problem containing Sulfur Dioxide at 41 degrees Kelvin, and 41 atmospheres of pressure. By entering the specs of the pipe diameter, length, friction factor, and mass flow rate, the calculator will return results in MATLAB containing the molar volume, volumetric flow rate, velocity of the compound, density, energy density, and the volumetric flow rate.

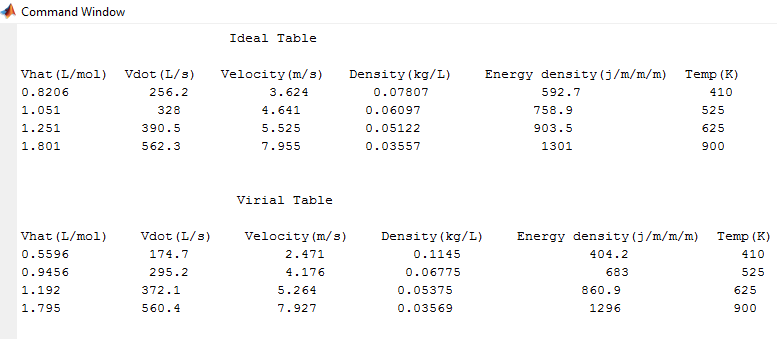
### Results

For this calculator, a few inputs are required in order to function properly. The user must input: what compound is being dealt with, pressure, temperature of the compound, diameter and length of the pipe, friction factor, and mass flow rate. The program will then extract specific data for the compound and use it in the computations for the resultant data.

Example:

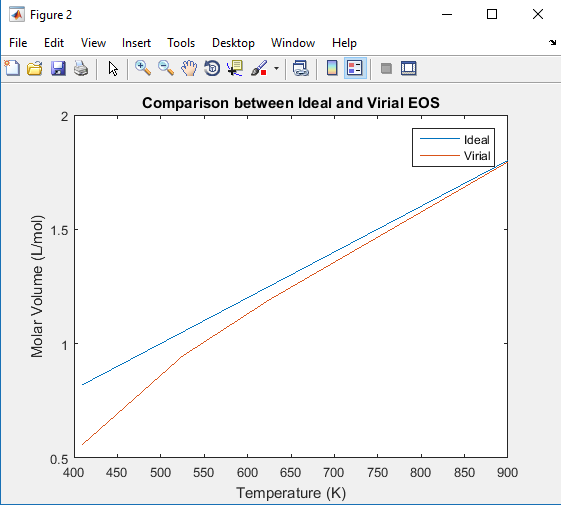
Say we want to find the molar volume and density of Carbon Dioxide at 525 degrees Kelvin and 4atm flowing through a 300 meter long pipe with a diameter of 0.4 meters. The pipe has a friction factor of 0.0003, and the mass flow rate of Carbon Dioxide is 30 kilograms per second.

The calculator will then extract the data for Carbon Dioxide from the data file (acentric factor, critical temperature, critical pressure, and molar mass). After extracting these values, the data will be input into the Ideal Gas Equation, as well as the Virial Equation. When completed, the program will return data on specific properties of the compound that is running through the system. The results will contain the molar volume, volumetric flow rate, velocity of the gas/fluid, energy density, and temperature of the compound to be analyzed.



**Figure 1.3**

This table is the data that is returned after using the calculator. The molar volume, volumetric flow rate, velocity, density, energy density, and temperature of the gas/fluid are displayed using both the Ideal Gas EOS and the Virial EOS.



**Figure 1.4**

After returning the resulting data, the calculator plots the temperature vs the molar volume when a temperature range is input. Note that the calculator can support more than five temperature data.

### Conclusion

With programs such as this, extremely intricate computations can be done accurately and efficiently in the matter of seconds. Material and Energy balances is one of the most complex subjects in chemical engineering, with this tool that complexity does not get in the way of getting an accurate answer. For students having the ability to do this, will benefit their ability to comprehend because they now have the ability to double check their answers and therefore go over their work. For professional engineers having a tool like this will enable, them to make their designs and see if the systems they are working with will be able to handle the current process specification and make necessary adjustments.

The program does have some limitations as the program does not take account of mass balances, but is based on estimation of only process variables. Use of a GUI creates the best level of user friendliness for the program, people whom are unfamiliar with Matlab will be able to easily use the program as each input has a specific place to input it, with the needed unit to properly run the program. As opposed to use of a function file which would require the user to know the function name, the number of outputs to define, and the specific number and order of inputs which in some cases the amount of data is large and this would make using functions very inconvenient. The GUI also allows for many functions and calculations to be run simultaneously without having to type in the inputs over and over again. This tool will prove to be very useful and will be a necessary tool for future Material and Energy Balance students.

### APPENDIX

|  |
| --- |
| Ammonia |
| Argon |
| Carbon Dioxide |
| Carbon Monoxide |
| Chlorine |
| Ethane |
| Hydrogen Sulfide |
| Methane |
| Methanol |
| Nitrogen |
| Oxygen |
| Propane |
| Sulfur Dioxide |
| Water |

**SUPPORTED GASES:**

SOURCE CODE:

GUI CONSTRUCTOR FUNCTIONS

function varargout = MEBcalc(varargin)  
% MEBCALC MATLAB code for MEBcalc.fig  
% MEBCALC, by itself, creates a new MEBCALC or raises the existing  
% singleton\*.  
%  
% H = MEBCALC returns the handle to a new MEBCALC or the handle to  
% the existing singleton\*.  
%  
% MEBCALC('CALLBACK',hObject,eventData,handles,...) calls the local  
% function named CALLBACK in MEBCALC.M with the given input arguments.  
%  
% MEBCALC('Property','Value',...) creates a new MEBCALC or raises the  
% existing singleton\*. Starting from the left, property value pairs are  
% applied to the GUI before MEBcalc\_OpeningFcn gets called. An  
% unrecognized property name or invalid value makes property application  
% stop. All inputs are passed to MEBcalc\_OpeningFcn via varargin.  
%  
% \*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one  
% instance to run (singleton)".  
%  
% See also: GUIDE, GUIDATA, GUIHANDLES  
  
% Edit the above text to modify the response to help MEBcalc  
  
% Last Modified by GUIDE v2.5 24-Nov-2015 23:10:50  
  
% Begin initialization code - DO NOT EDIT  
gui\_Singleton = 1;  
gui\_State = struct('gui\_Name', mfilename, ...  
 'gui\_Singleton', gui\_Singleton, ...  
 'gui\_OpeningFcn', @MEBcalc\_OpeningFcn, ...  
 'gui\_OutputFcn', @MEBcalc\_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 MEBcalc is made visible.  
function MEBcalc\_OpeningFcn(hObject, eventdata, handles, varargin)  
% This function has no output args, see OutputFcn.  
% hObject handle to figure  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles structure with handles and user data (see GUIDATA)  
% varargin command line arguments to MEBcalc (see VARARGIN)  
  
% Choose default command line output for MEBcalc  
handles.output = hObject;  
  
% Update handles structure  
guidata(hObject, handles);  
  
% UIWAIT makes MEBcalc wait for user response (see UIRESUME)  
% uiwait(handles.figure1);  
  
  
% --- Outputs from this function are returned to the command line.  
function varargout = MEBcalc\_OutputFcn(hObject, eventdata, handles)  
% varargout cell array for returning output args (see VARARGOUT);  
% hObject handle to figure  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles structure with handles and user data (see GUIDATA)  
  
% Get default command line output from handles structure  
varargout{1} = handles.output;  
  
  
% --- Executes on button press in Compute.  
function Compute\_Callback(hObject, eventdata, handles)  
% hObject handle to Compute (see GCBO)  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles structure with handles and user data (see GUIDATA)  
  
% THESE BLOCKS OF CODES HANDLES DATA EXTRACTION  
% CALLS TO FUNCTIONS  
 handles.data = data\_extractor(handles.comp);  
 pa\_factor = handles.data(1,1);  
 mw = handles.data(1,2);  
 ctemp = handles.data(1,3);  
 cpress = handles.data(1,4);  
 L = handles.lendata(1,1);  
 D = handles.lendata(1,2);  
 K = handles.lendata(1,3);  
  
 EOS2(handles.press,handles.temp,handles.mass,L,D,K,pa\_factor,ctemp,cpress,mw);  
  
 guidata(hObject,handles);  
  
  
  
function temp\_text\_Callback(hObject, eventdata, handles)  
% hObject handle to temp\_text (see GCBO)  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles structure with handles and user data (see GUIDATA)  
  
% FOR LOOP STRUCTURE TO CHECK WHETHER THE TEMPERATURE DATA  
% IS COMMA SEPARATED OR SPACE SEPARATED  
x = 0;  
for i = get(hObject, 'String')  
  
 if i == ','  
 temparray = strsplit(get(hObject, 'String'),',');  
 x = x + 1;  
 break  
 elseif i == ' '  
 temparray = strsplit(get(hObject, 'String'),' ');  
 x = x + 1;  
 break  
 end  
  
end  
  
if x == 0  
 temparray(1) = str2double(get(hObject, 'String'));  
end  
  
% THIS BLOCK OF CODE CHECKS THE NUMBER OF TEMPERATURE DATA ENTERED  
% AND PUT THEM INTO AN ARRAY OF CORREPONDING SIZE  
if length(temparray) == 1  
 if isa(temparray(1,1),'double') == 0  
 temp1 = str2double(temparray(1,1));  
 tempA = [temp1];  
 else  
 tempA = [temparray(1,1)];  
 end  
  
else  
 tempA = rand(1,numel(temparray));  
 for i = 1:numel(temparray)  
 tempA(i) = str2double(temparray(i));  
 end  
  
end  
  
  
handles.temp = tempA; % Saves data to GUI structure  
guidata(hObject, handles); % saves data into the GUI data stack  
% Hints: get(hObject,'String') returns contents of temp\_text as text  
% str2double(get(hObject,'String')) returns contents of temp\_text as a double  
  
  
% --- Executes during object creation, after setting all properties.  
function temp\_text\_CreateFcn(hObject, eventdata, handles)  
% hObject handle to temp\_text (see GCBO)  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles empty - handles not created until after all CreateFcns called  
  
% Hint: edit controls usually have a white background on Windows.  
% See ISPC and COMPUTER.  
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))  
 set(hObject,'BackgroundColor','white');  
end  
  
  
  
function pressure\_text\_Callback(hObject, eventdata, handles)  
% hObject handle to pressure\_text (see GCBO)  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles structure with handles and user data (see GUIDATA)  
handles.press = str2double(get(hObject, 'String')); %gets the pressure  
guidata(hObject, handles);  
% Hints: get(hObject,'String') returns contents of pressure\_text as text  
% str2double(get(hObject,'String')) returns contents of pressure\_text as a double  
  
  
% --- Executes during object creation, after setting all properties.  
function pressure\_text\_CreateFcn(hObject, eventdata, handles)  
% hObject handle to pressure\_text (see GCBO)  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles empty - handles not created until after all CreateFcns called  
  
% Hint: edit controls usually have a white background on Windows.  
% See ISPC and COMPUTER.  
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))  
 set(hObject,'BackgroundColor','white');  
end  
  
  
  
function Mass\_flow\_text\_Callback(hObject, eventdata, handles)  
% hObject handle to Mass\_flow\_text (see GCBO)  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles structure with handles and user data (see GUIDATA)  
handles.mass = str2double(get(hObject, 'String')); % Gets mass data  
guidata(hObject, handles);  
% Hints: get(hObject,'String') returns contents of Mass\_flow\_text as text  
% str2double(get(hObject,'String')) returns contents of Mass\_flow\_text as a double  
  
  
% --- Executes during object creation, after setting all properties.  
function Mass\_flow\_text\_CreateFcn(hObject, eventdata, handles)  
% hObject handle to Mass\_flow\_text (see GCBO)  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles empty - handles not created until after all CreateFcns called  
  
% Hint: edit controls usually have a white background on Windows.  
% See ISPC and COMPUTER.  
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))  
 set(hObject,'BackgroundColor','white');  
end  
  
  
  
function comp\_text\_Callback(hObject, eventdata, handles)  
% hObject handle to comp\_text (see GCBO)  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles structure with handles and user data (see GUIDATA)  
handles.comp = get(hObject,'String'); % gets compound name  
guidata(hObject, handles); %  
% Hints: get(hObject,'String') returns contents of comp\_text as text  
% str2double(get(hObject,'String')) returns contents of comp\_text as a double  
  
  
% --- Executes during object creation, after setting all properties.  
function comp\_text\_CreateFcn(hObject, eventdata, handles)  
% hObject handle to comp\_text (see GCBO)  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles empty - handles not created until after all CreateFcns called  
  
% Hint: edit controls usually have a white background on Windows.  
% See ISPC and COMPUTER.  
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))  
 set(hObject,'BackgroundColor','white');  
end  
  
  
  
function length\_text\_Callback(hObject, eventdata, handles)  
% hObject handle to length\_text (see GCBO)  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles structure with handles and user data (see GUIDATA)  
  
% Array parsing  
% This block of code gets, and parses data from the text box  
lendata = strsplit(get(hObject, 'String'),',');  
L = str2double(lendata(1,1));  
D = str2double(lendata(1,2));  
K = str2double(lendata(1,3));  
lendata = [L D K];  
handles.lendata = lendata;  
guidata(hObject, handles);  
  
% Hints: get(hObject,'String') returns contents of length\_text as text  
% str2double(get(hObject,'String')) returns contents of length\_text as a double  
  
  
% --- Executes during object creation, after setting all properties.  
function length\_text\_CreateFcn(hObject, eventdata, handles)  
% hObject handle to length\_text (see GCBO)  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles empty - handles not created until after all CreateFcns called  
  
% Hint: edit controls usually have a white background on Windows.  
% See ISPC and COMPUTER.  
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))  
 set(hObject,'BackgroundColor','white');  
end  
  
  
  
function x\_Callback(hObject, eventdata, handles)  
% hObject handle to x (see GCBO)  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles structure with handles and user data (see GUIDATA)  
handles.interpdataXk = str2double(get(hObject, 'String'));  
guidata(hObject, handles);  
% Hints: get(hObject,'String') returns contents of x as text  
% str2double(get(hObject,'String')) returns contents of x as a double  
  
  
% --- Executes during object creation, after setting all properties.  
function x\_CreateFcn(hObject, eventdata, handles)  
% hObject handle to x (see GCBO)  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles empty - handles not created until after all CreateFcns called  
  
% Hint: edit controls usually have a white background on Windows.  
% See ISPC and COMPUTER.  
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))  
 set(hObject,'BackgroundColor','white');  
end  
  
  
  
function x1\_Callback(hObject, eventdata, handles)  
% hObject handle to x1 (see GCBO)  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles structure with handles and user data (see GUIDATA)  
handles.interpdataX(1) = str2double(get(hObject, 'String'));  
guidata(hObject, handles);  
% Hints: get(hObject,'String') returns contents of x1 as text  
% str2double(get(hObject,'String')) returns contents of x1 as a double  
  
  
% --- Executes during object creation, after setting all properties.  
function x1\_CreateFcn(hObject, eventdata, handles)  
% hObject handle to x1 (see GCBO)  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles empty - handles not created until after all CreateFcns called  
  
% Hint: edit controls usually have a white background on Windows.  
% See ISPC and COMPUTER.  
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))  
 set(hObject,'BackgroundColor','white');  
end  
  
  
  
function x2\_Callback(hObject, eventdata, handles)  
% hObject handle to x2 (see GCBO)  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles structure with handles and user data (see GUIDATA)  
handles.interpdataX(2) = str2double(get(hObject, 'String'));  
guidata(hObject, handles);  
% Hints: get(hObject,'String') returns contents of x2 as text  
% str2double(get(hObject,'String')) returns contents of x2 as a double  
  
  
% --- Executes during object creation, after setting all properties.  
function x2\_CreateFcn(hObject, eventdata, handles)  
% hObject handle to x2 (see GCBO)  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles empty - handles not created until after all CreateFcns called  
  
% Hint: edit controls usually have a white background on Windows.  
% See ISPC and COMPUTER.  
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))  
 set(hObject,'BackgroundColor','white');  
end  
  
  
  
function y1\_Callback(hObject, eventdata, handles)  
% hObject handle to y1 (see GCBO)  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles structure with handles and user data (see GUIDATA)  
handles.interpdataY(1) = str2double(get(hObject, 'String'));  
guidata(hObject, handles);  
% Hints: get(hObject,'String') returns contents of y1 as text  
% str2double(get(hObject,'String')) returns contents of y1 as a double  
  
  
% --- Executes during object creation, after setting all properties.  
function y1\_CreateFcn(hObject, eventdata, handles)  
% hObject handle to y1 (see GCBO)  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles empty - handles not created until after all CreateFcns called  
  
% Hint: edit controls usually have a white background on Windows.  
% See ISPC and COMPUTER.  
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))  
 set(hObject,'BackgroundColor','white');  
end  
  
  
  
function y2\_Callback(hObject, eventdata, handles)  
% hObject handle to y2 (see GCBO)  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles structure with handles and user data (see GUIDATA)  
handles.interpdataY(2) = str2double(get(hObject, 'String'));  
guidata(hObject, handles);  
  
% Hints: get(hObject,'String') returns contents of y2 as text  
% str2double(get(hObject,'String')) returns contents of y2 as a double  
  
  
% --- Executes during object creation, after setting all properties.  
function y2\_CreateFcn(hObject, eventdata, handles)  
% hObject handle to y2 (see GCBO)  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles empty - handles not created until after all CreateFcns called  
  
% Hint: edit controls usually have a white background on Windows.  
% See ISPC and COMPUTER.  
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))  
 set(hObject,'BackgroundColor','white');  
end  
  
  
  
function y\_text\_Callback(hObject, eventdata, handles)  
% hObject handle to y\_text (see GCBO)  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles structure with handles and user data (see GUIDATA)  
% y = handles.y;  
  
% Hints: get(hObject,'String') returns contents of y\_text as text  
% str2double(get(hObject,'String')) returns contents of y\_text as a double  
  
  
% --- Executes during object creation, after setting all properties.  
function y\_text\_CreateFcn(hObject, eventdata, handles)  
% hObject handle to y\_text (see GCBO)  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles empty - handles not created until after all CreateFcns called  
  
% Hint: edit controls usually have a white background on Windows.  
% See ISPC and COMPUTER.  
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))  
 set(hObject,'BackgroundColor','white');  
end  
  
  
% --- Executes on button press in interpolator.  
function interpolator\_Callback(hObject, eventdata, handles)  
% hObject handle to interpolator (see GCBO)  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles structure with handles and user data (see GUIDATA)  
x = handles.interpdataX;  
y = handles.interpdataY;  
xk = handles.interpdataXk;  
  
y = interpo(x,y,xk);  
y = num2str(y);  
set(handles.Ytext,'String', y);  
  
guidata(hObject, handles);

Data\_Extractor function

function [ Data ] = data\_extractor( strg )  
%Takes a string args and parses it to retrieve the correct data from data  
%table  
% tailed explanation goes here  
data = importdata('Project\_Data.xlsx');  
names = data.textdata;  
names = names([2:end],1);  
data = data.data;  
  
  
if length(strg) == length('Ammonia') & strg == 'Ammonia'  
 Data = data(1,:);  
  
elseif length(strg) == length('Argon') & strg == 'Argon'  
 Data = data(2,:);  
  
elseif length(strg) == length('Carbon Dioxide') & strg == 'Carbon Dioxide'  
 Data = data(3,:);  
  
elseif length(strg) == length('Carbon Monoxide') & strg == 'Carbon Monoxide'  
 Data = data(4,:);  
  
elseif length(strg) == length('Chlorine') & strg == 'Chlorine'  
 Data = data(5,:);  
  
elseif length(strg) == length('Ethane') & strg == 'Ethane'  
 Data = data(6,:);  
  
elseif length(strg) == length('Hydrogen Sulfide') & strg == 'Hydrogen Sulfide'  
 Data = data(7,:);  
  
elseif length(strg) == length('Methane') & strg == 'Methane'  
 Data = data(8,:);  
  
elseif length(strg) == length('Methanol') & strg == 'Methanol'  
 Data = data(9,:);  
  
elseif length(strg) == length('Nitrogen') & strg == 'Nitrogen'  
 Data = data(10,:);  
  
elseif length(strg) == length('Oxygen') & strg == 'Oxygen'  
 Data = data(11,:);  
  
elseif length(strg) == length('Propane') & strg == 'Propane'  
 Data = data(12,:);  
  
elseif length(strg) == length('Sulfur Dioxide') & strg == 'Sulfur Dioxide'  
 Data = data(13,:);  
  
elseif length(strg) == length('Water') & strg == 'Water'  
 Data = data(14,:);  
  
else  
 Data = 'Element/Compound not found.'  
  
end  
  
  
  
  
end

EOS2 FUNCTION

%AUTHOR: UME-UGWA CHUKWUBUIKEM  
function EOS2( Pressure, Temperature, mass,L, D, K, pA, cT, cP, mW )  
% Uses different equation of states to calculate volumetric flow rate, molar  
% volume flow rate, and other variables given temperature, pressure and mass flow rate  
% Constants are define within the function and other system variables such  
% as length, diameter etc  
%  
  
%INITIALIZE VARIABLES  
 P = Pressure; % Pressure in (atm)  
 T = Temperature; % Temperature in (K)  
 m = mass; % Mass flow rate in (kg/s)  
 % L = L; % Length of the pipe in (ft)  
 % D = D; % Diameter of the pipe in (inch)  
 f = K; % Given constant for energy density calculation  
 R = 0.08206; % Universal gas constant in (atm\*l/mol\*k)  
 Mw = mW; % molecular weight of SO2 in (kg/kmol)  
  
 % mole calculation for SO2  
 n = (m/Mw)\*1000; % molar flow rate of SO2 in (mol/s) given the mass flow rate  
  
 % Converts length and diameter to meters  
% L = L/3.2808; % Length in (m)  
% D = D/39.37; % Diameter in (m)  
  
 % Calculates area  
 A = pi\*((D^2)/4); % Cross-sectional area in (m^2)  
  
 function [Itable] = idealGas()  
 % Computes volumtric flow rate, molar volume and other required variables  
 % using the ideal gas EOS  
  
 Vflow\_I = (n\*R.\*T)./P; % Volumeric flow rate in (L/s)  
  
 Vhat\_I = Vflow\_I./n; % Molar Volume in (L/mol)  
  
 velocity\_I = (Vflow\_I/1000)./A; % Velocity of gas in (m/s)  
  
 density\_I = m./Vflow\_I; % Density of gas in (kg/L) equivalent to (gm/cc)  
  
 uLoss\_I = 4\*f\*(L/D)\*0.5\*(density\_I\*1000).\*velocity\_I.^2; % Loss estimate in (Pa\*m)  
  
 % DATA TABLE CREATION  
 idealTable = [Vhat\_I; Vflow\_I; velocity\_I; density\_I; uLoss\_I;T];  
 Itable = Vhat\_I;  
 disp(' Ideal Table')  
 fprintf('\n')  
 disp('Vhat(L/mol) Vdot(L/s) Velocity(m/s) Density(kg/L) Energy density(j/m/m/m) Temp(K)')  
 fprintf('%4.4g %14.4g %14.4g %14.4g %18.4g %18.4g\n', idealTable)  
 fprintf('\n\n')  
  
 end  
  
 function [Vtable] = virialTrunc()  
 %Uses the virial truncated EOS to calculate for the requested system  
 %variables  
  
 % Estimation of virial two body interaction constant  
  
 w = pA; % pitzer acentric factor for SO2  
 Tc = cT; % critical temperature (K)  
 Pc = cP; % critical pressusre (atm)  
  
 Pr = P / Pc; % Reduced pressure to find z from the chart  
 Tr = T / Tc; % Reduced temperature to find z from the chart  
  
 Bo = 0.083 - (0.422 ./ (Tr.^1.6)); % calculates Bo for estimation of B  
 B1 = 0.139 - (0.172 ./ (Tr.^4.2)); % calculates B1 for estimation of B  
  
 B = ((R \* Tc) / Pc) \* (Bo + w \* B1); % Calculates virial 2body constant  
  
 % Vhat CALCULATION  
 '(P/RT)\*Vhat^2 - Vhat - B = 0'; % Derived quadratic equation from virial truncated EOS  
  
 Vhat\_V = rand(1,numel(T));  
  
 for i = 1:numel(T)  
  
 a = P/(R\*T(i)); % Coefficient of Vhat^2 from the quadratic  
 b = -1; % Coefficient of Vhat from the quadratic  
 c = -1\*B(i); % Virial two body interaction constant from the quadratic  
 k1 = [a b c]; % Array of coefficients of vhat and constant B at the given temperature  
 r = roots(k1); % Finds the roots of the quadratic at the given temperature  
 Vhat\_V(i) = r(1);  
  
  
 end  
  
  
 Vflow\_V = Vhat\_V\*n; % Volumetric flow rate in (L/s)  
  
 velocity\_V = (Vflow\_V/1000)/A; % velocity in (m/s)  
  
 density\_V = m./Vflow\_V; % Density in (kg/L) equivalent to (gm/cc)  
  
 uLoss\_V = 4\*f\*(L/D)\*.5\*(density\_V\*1000).\*velocity\_V.^2; % Loss estimate in (j/m^3)  
  
 % DATA TABLE CREATION  
 VirialTable = [Vhat\_V; Vflow\_V; velocity\_V; density\_V; uLoss\_V; T];  
 disp(' Virial Table')  
 fprintf('\n')  
 disp('Vhat(L/mol) Vdot(L/s) Velocity(m/s) Density(kg/L) Energy density(j/m/m/m) Temp(K)')  
 fprintf('%4.4g %14.4g %16.4g %16.4g %20.4g %16.4g\n', VirialTable)  
 fprintf('\n')  
 Vtable = Vhat\_V;  
  
 end  
 [Itable] = idealGas; % Calls the ideal gas EOS function to calculate the required variables  
 [Vtable] = virialTrunc; % Calls the virial truncated EOS function to calculate the required variables  
 figure  
 plot(T,Itable)  
 hold on  
 plot(T,Vtable)  
 title('Comparison between Ideal and Virial EOS')  
 ylabel('Molar Volume (L/mol)')  
 xlabel('Temperature (K)')  
 legend('Ideal','Virial','BestOutside')  
  
end

[*Published with MATLAB® R2015a*](http://www.mathworks.com/products/matlab)