MATLAB Workshop 10 - Decision Making: menus and **switch/case**

Objectives: Learn two methods to display a menu, use of a **while** loop to perform error checking for input information, use of the **switch/case** construction to make decisions. Learn to break a large script into smaller functions and design and test each function separately.

MATLAB Features:

relational operators

Symbol	Meaning
==	is the same as
~=	is not the same as
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

logical operators

Symbol	Meaning
&	logical and
|	logical or
~	logical not

workspace commands

Command	Action
pause	halts command/window/script/function execution pending response
menu	creates pop-up window with choices as buttons

repetitive action **while** *loop structure*

```
loop body initialization statements
loop condition initialization
while (condition is true)
    execute loop body
    update to see whether condition is now false
end  % while
```

multiple selection ***switch/case*** *structure*

```
switch (expression_value)
case  {vA1, vA2}  % execute when expression_value == vA1 or vA2
     action A1
     action A2
     ...
case  {vB1, vB2}  % execute when expression_value == vB1 or vB2
     action B1
     action B2
     ...
...
otherwise  % optional default if expression_value ≠ any prior value
     action default1
     action default2
     ...
end
```

- **A thermodynamics problem**

Gas phase thermodynamic equations of state relate the three state variables of temperature, pressure, and volume for a gas. One of the three state variables can be calculated through the equation of state if values for the other two variables are known. For example, the ideal gas law states

$$P\tilde{V} = RT$$

where

P	:	pressure, Pa
$\tilde{V}$	:	specific or molar gas volume, m^3mol
R	:	ideal gas constant, (= 8.314 J/(mol K))
T	:	absolute temperature, K

Two other popular equations of state are the van der Waals equation of state given by

$$\left(P + \frac{a}{\tilde{V}^2}\right)(\tilde{V} - b) = RT$$

where *a* and *b* are constants that depend upon the particular gas. Oxygen will have different values for *a* and *b* than carbon dioxide. Can you figure out what the units associated with *a* and *b* are, respectively? The Redlich-Kwong equation of state is more complex and is given by

$$P = \frac{RT}{\tilde{V} - b} - \frac{a}{T^{0.5}\tilde{V}(\tilde{V} + b)}$$

where *a* and *b* are, again, constants associated with a particular gas. Can you figure out what the units associated with *a* and *b* are, respectively, for the Redlich-Kwong equation of state? Note that even though the same symbols, *a* and *b*, are used in both the van der Waals and Redlich-Kwong equations of state, the symbols do not have the same meaning.

Your thermodynamics instructor has asked you to create a MATLAB script that will calculate the pressure for specified temperature and specific volume of any gas by any of the three equations of state.

- **Developing a plan of attack**

You should develop a coherent plan of attack (a program outline) before trying to actually code a new script. This requires that you do some background thinking and scratching on a piece of paper before you touch the computer keyboard. A small amount of time invested at this stage will save hours of frustration at the computer keyboard. You need to remember: **computers are stupid!** They will do exactly what you tell them to do. But, if you cannot tell them exactly what to do (step-by-step), they will complain and fuss at you and frustrate you.

A good way to develop a plan of attack is to divide (and conquer!) a large problem into smaller components. Properly combining solutions for the smaller components in the proper order should yield a solution to the large problem. Most computer problems can be broken down into the distinctly separate actions of

1. getting information into the computer
2. using the information to create new information (computations)
3. reporting the results of the new information generated.

Each action may require one or more task (function) to accomplish the action. The script design for the actions does not need to be done in a linear fashion.

As an illustration, the computation section of the equation of state problem can be developed as

```
function  pcalc
```

(1) Function to calculate the pressure using the (1) ideal gas law, (2) van der Waals equation, or (3) Redlich-Kwong equation.
(2) Produces **`pressure`** (in SI units).
(3) Needs
    `eqn_choice` (1 = ideal gas, 2 = van der Waals, 3 = Redlich-Kwong)
    `temp` (temperature in SI units)
    `sp_vol` (specific volume in SI units)
    a (vdW or RK parameter)
    b (vdW or RK parameter)

Note: R is a constant that is embedded in the function. It is not an information "need".
(4) Algorithm

```
Need to make a decision based on the value of eqn_choice
    if ( eqn_choice is 1 )
        use ideal gas law to calculate pressure
    elseif ( eqn_choice is 2 )
        use vdW equation to calculate pressure
    else (Note: by default eqn_choice is 3 ==> else rather than elseif )
        use RK equation to calculate pressure
```

Developing the computation section first tells us that we need to have values for **`eqn_choice`**, **`temp`**, **`sp_vol`**, **a**, and **b** available in the computer before the function can be called. The only way to get these values into the computer is through action 1: getting information into the computer. So we turn our attention to action 1.

`function pcalc_get_info`

(1) Function to get equation choice (1) ideal gas law, (2) van der Waals equation, or (3) Redlich-Kwong equation, and temperature, specific volume, and vdW or RK constants from user.
(2) Produces
    `eqn_choice` (1 = ideal gas, 2 = van der Waals, 3 = Redlich-Kwong)
    `temp` (temperature in SI units)
    `sp_vol` (specific volume in SI units)
    a (vdW or RK parameter)
    b (vdW or RK parameter)
(3) **Needs nothing**!!! Remember, this is asking what information the function needs that is already *inside* the computer. This function will get its information from *outside* the computer.
(4) Algorithm

```
Ask user for eqn_choice  (1 = ideal gas, 2 = van der Waals, 3 = Redlich-Kwong)
Get temp (temperature in SI units)
Get sp_vol (specific volume in SI units)
Get a and b
    if ( eqn_choice is 1 )
        Set a and b to zero (not needed for ideal gas law)
    elseif ( eqn_choice is 2 )
        Identify eqn of state = vdW
        Get a and b (be sure to note units when asking)
    else (Note: by default eqn_choice is 3 ==> else rather than elseif )
        Identify eqn of state = RK
        Get a and b (be sure to note units when asking)
```

Finally, the output section can be developed. The major effort in designing the output section should be to ensure that the resulting display is user-friendly and easy to read.

function pcalc_results

(1) Display pressure for given temperature and specific volume as calculated by chosen equation of state (and appropriate parameters).

(2) **Produces nothing**!!! Remember, this is asking what information the function will return *inside* the computer. This function displays results *outside* the computer.

(3) Needs **eqn_choice** (1 = ideal gas, 2 = van der Waals, 3 = Redlich-Kwong)
pressure (pressure in SI units)
temp (temperature in SI units)
sp_vol (specific volume in SI units)
a (vdW or RK parameter)
b (vdW or RK parameter)

(4) Algorithm

```
Display pressure  (with SI units)
Display temp (with SI units)
Display sp_vol (with SI units)
Display eqn of state and a and b
    if ( eqn_choice is 1 )
        Display ideal gas law
    elseif ( eqn_choice is 2 )
        Display vdW equation of state
        Display a and b (with SI units)
    else (Note: by default eqn_choice is 3 ==> else rather than elseif )
        Display RK equation of state
        Display a and b (with SI units)
```

Putting these three functions back together should produce the desired script. Note how the information flows from one function to another.

- **Asking the user to choose - *menus***

A common programming practice is to ask the user to select a choice from a list of options (or menu) for which set of actions to perform. The function or script then takes actions appropriate to the choice entered by the user. MATLAB provides two methods for acquiring a user response.

(1) Using **disp** and **input** to create a menu.

We have already used **disp** to display text messages in the Command Window and **input** to get values from the user. A simple menu is easily created by combining these two commands. **disp** is used to display the options. **input** is used to get the choice.

Open a new file in the MATLAB editor and enter the following.

```
function  [eqn_choice, temp, sp_vol, a, b ] = pcalc_get_info()
%  function header information

% variable dictionary
%

% get user choice for equation of state
    disp(' ')
disp('What equation of state would you like?')
disp('   (1) Ideal gas law')
disp('   (2) Van der Waals')
disp('   (3) Redlich-Kwong')
eqn_choice = input('Please enter choice (1, 2, 3) ==> ');
disp(' ')
```

Be sure to enter appropriate code for the function header information and variable dictionary sections.

disp commands are used to display textual information. The **input** command is used to pick up the user response.

Style notes
- Notice the indenting. Indenting is used to identify sections of code that belong together.
- A blank **disp(' ')** before and after text display separates the text in the Command Window when it runs. This is user-friendly and makes your scripts more pleasant to use.
- Note the indenting inside the **disp** commands. This also makes your display more user-friendly.

Save the Editor window as **pcalc_get_info.m**. Test your function so far by entering

```
» pcalc_get_info
```

at the Command prompt (**Note**: you can test partially completed functions!!!). The menu above should appear. Respond with a 1. What value displays for **ans**? (Remember the first value in the output list is assigned to **ans**). So far, so good. Now run the function two more times, answering first 2 and then 3. What happens if you respond with a 4? **Is this good?**

(2) Error checking input.

A major problem with menus is assuring that the user response is actually one of the permitted responses (remember, you are building your script under the assumption that valid input is available later on). Because users will tend to mess up your program whenever possible, you need to limit that potential. The following code will provide an error check to assure that only an allowed value is entered.

```
% get user choice for equation of state
     errorflag = 0;
     while (errorflag == 0)
          disp(' ')
          disp('What equation of state would you like?')
          disp('   (1) Ideal gas law')
          disp('   (2) Van der Waals')
          disp('   (3) Redlich-Kwong')
          eqn_choice = input('Please enter choice (1, 2, 3) ==> ');
          disp(' ')
          % check for valid input
               if( eqn_choice == 1 | eqn_choice == 2 | eqn_choice == 3)
                    errorflag ==1;
               else
                    disp('   *** Invalid choice ')
                    disp('   *** Please respond with a 1, 2, or 3 ')
                    disp('   *** Press ENTER to continue ' )
                    pause
               end  % if
     end  % while
```

This code uses a **while** loop to assure the input of valid information. The general form for a while loop is

```
loop body initialization statements
loop condition initialization
while (condition is true)
    execute loop body
    update to see whether condition is now false
end  % while
```

In the above code, **errorflag** is initialized and used for the *condition*, the **disp** and **input** commands of the menu (which do not need to be initialized) form the *loop body*, and the **if/else** is the *update* to change the condition (**errorflag**) if valid input has been obtained. If a valid choice has not been obtained, an error message is displayed and the loop recycles. If a valid choice has been entered, the condition becomes false and the loop is exited.

The above code also uses a new MATLAB command, **pause**. **pause** causes a script or function to stop and wait for the user to respond (by hitting any key, not just the enter key) before continuing. The use of **pause** here is user friendly because it forces the user to respond to the fact that an error has taken place before redisplaying the menu. Try running the script with and without the **pause**. Which do you find better?

Finally, a new variable, **errorflag**, was defined. Did you add it to the variable dictionary?

(3) Using **menu** to create a menu.

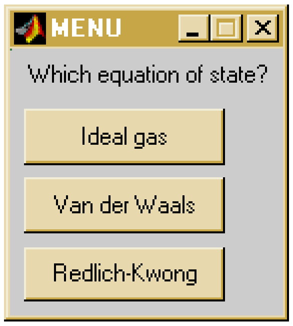
MATLAB also has a built-in **menu** function that is convenient to use. It creates a pop-up window with buttons for the response. The general format for the menu command is

```
response = menu('header', 'item1', 'item2', ...  );
```

MATLAB will place the **header** at the top of the menu pop-up window and **item1**, **item2**, etc, will be on the window buttons. When the user clicks on one of the buttons, the button number (1 for **item1**, 2 for **item2**, etc) will be assigned to **response**. **response** can be any descriptive name. To see how this works, enter the following into the MATLAB Command Window

```
» eqn_choice = menu('Which equation of state?', 'Ideal gas', ...
'Van der Waals', 'Redlich-Kwong');
```

The pop-up menu window depicted at the right should appear. Click on one of the buttons and the window disappears. Note that a semicolon was used in the menu command to suppress display of the value assigned to **eqn_choice** (there is no need for the user to know your secret codes!) Check to see what value was assigned by typing



```
» eqn_choice
```

MATLAB should respond with a 1 if you clicked on *Ideal gas*, 2 if you clicked on *Van der Waals*, or 3 if you clicked on *Redlich-Kwong*.

The **menu** command can replace the entire text-based menu above. There is no need to have error checking because the user can only select one of the buttons and MATLAB assigns the integers 1, 2, ... in order to the buttons.

If you want, you can replace the text-based menu with the menu command. If you do, be sure to remove **errorflag** from the variable dictionary.

(4) Using **switch/case** to make decisions.

MATLAB has an alternative method to the **if/elseif/else** structure to evaluate and execute decisions, as shown here:

```
switch (expression_value)
case  valueA  % execute when expression_value == valueA
    action A1
    action A2
    ...
case  valueB  % execute when expression_value == valueB
    action B1
    action B2
    ...
...
otherwise  % optional default if expression_value ≠ any value
    action default1
    action default2
    ...
end
```

If the same actions are to be taken for more than one value of **expression_value**, which is equivalent to a *logical or*, i.e., A or B, the switch/case construct is slightly altered to

```
switch (expression_value)
case  {vA1, vA2}  % expression_value == vA1 or vA2
    action A1
    action A2
    ...
case  {vB1, vB2}  % expression_value == vB1 or vB2
    action B1
    action B2
    ...
...
otherwise  % optional default if expression_value ≠ any value
    action default1
    action default2
    ...
end
```

In contrast with the **if/elseif/else** construct, which tests for the truth of a logical expression, the **switch/case** construct simply compares the results of evaluating any legitimate MATLAB expression that produces a value with a list of possible outcomes (cases). As we have seen, a legitimate MATLAB expression can be simply a "hard-wired" value, a variable name (which is a command to get the variable value), an arithmetic expression, or an algebraic expression that includes variable names and functions. If a match is made between the resulting **expression_value** and any of the **case values**, the actions associated with the matching case are executed. If no matches are made, then no actions are taken unless the optional **otherwise** clause is included.

Note: because the **switch/case** is testing for equivalency of values, only expressions that produce integers or characters can be used with the **switch/case**. Strange and unexpected results can and will occur if an expression that produces a decimal number is used.

Let's complete the coding for **function pcalc_get_info**. Following the menu code for getting **eqn_choice**, add the following

```
% get temp and specific volume
     temp = input('Please enter temperature in K ==> ' );
     sp_vol = input('Please enter specific volume in m^3/mol ==> ' );

% get van der Waals or Redlich-Kwong parameters
     if (eqn_choice == 1)  % ideal gas law
          a = 0;
          b = 0;
     elseif (eqn_choice == 2 ) % van der Waals
          a = input('Please enter a in ??? ==> ' );
          b = input('Please enter b in ??? ==> ' );
     else  % must be Redlich-Kwong
          a = input('Please enter a in ??? ==> ' );
          b = input('Please enter b in ??? ==> ' );
     end
```

Save the function and test it by entering

```
» [eqn_choice, temp, sp_vol, a, b] = pcalc_get_info
```

at the Command prompt. Since no semicolon was used to suppress display, MATLAB should respond with the values you entered.

Change the **if/elseif/else** structure to a **switch/case** structure, save the function, and retest it. Do you get the same results?

(5) Create **function pcalc**.

Encode **function pcalc** (see design above to refresh your memory) in the MATLAB Editor. Be sure to have a function header section and variable dictionary. Use an **if/elseif/else** structure to pick the appropriate equation of state and subsequent calculations. Test your function (How? By calling it in the Command Window with values for which you know the results).

(6) Create **function pcalc_results**.

Encode **function pcalc_results** (see design above to refresh your memory) in the MATLAB Editor. Be sure to have a function header section and variable dictionary. Use a

switch/case structure to display the appropriate equation of state and subsequent information. Test your function (How? By calling it in the Command Window with values for which you know the results).

(7) Create the script **wkshp10**.

You now have the requisite functions to create a script that puts it all together. Create a script, **wkshp10.m**, that does the following

- displays script header information (call to **wkshp10_header**)
- gets the necessary input information (call to **pcalc_get_info**)
- performs the pressure calculation (call to **pcalc**)
- displays the results (call to **pcalc_results**)

Be sure your script has an appropriate header section and variable dictionary.

Bonus question: Can you add a **while** loop to your script so that the script asks the user whether to repeat for another pressure calculation and, if the answer is yes, repeats the input, calculation, display portion of the script? (You do not generally want to display the header section each time the script repeats - once is enough).

Recap: You should have learned

- The relational operators in MATLAB.
- The logical operators in MATLAB.
- The utility of breaking a larger script into smaller tasks.
- The utility of outlining a function before coding.
- Two methods of creating menus.
- How **pause** can be used to halt script/function execution until a response is made.
- How the **menu** command creates a pop-up window with buttons.
- How a **while** loop operates.
- The multiple condition **switch/case** structure.
- That the **switch/case** structure should only be used for integer or character **switch** values.
- To test your scripts/functions with known values before trusting them to solve properly solve unknown problems.