**Lab Workshop #9**

Purpose: incorporating partial pivoting into the Gaussian Elimination algorithm

using message boxes with their options

adding user forms for dedicated user interfaces

1. Launch Excel 2007 and open **Gauss.xlsm**.

2. On the spreadsheet, change the coefficients so that they appear as



Run the Gauss macro (use the Developer 🡺 Macro 🡺 Run). What do you observe?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

A limitation of this first-try Gaussian elimination algorithm is that it will fail if any pivot element becomes zero during the calculation. This occurs for the example with the 2,2 pivot. But this set of equations does have a solution, so it is unacceptable for our algorithm to fail here.

The strategy employed to remedy this limitation is called "partial pivoting." The concept is easy to explain in words: When we encounter a pivot element that is zero (or really small), we look at the elements directly below the pivot element and find the largest one (absolute value). We then swap the pivot row with the row with the largest value. This is like swapping two equations – it doesn't change the solution. But it does provide us with a non-zero (non-small) pivot element, and the algorithm can proceed. Note: if there is no element below the pivot that is not zero (or not small), that does mean that the set of equations does not have a solution. Another word for this is that the equation set is singular.

So, the flowchart below is a modification of part of the previous flowchart for the Gaussian Elimination algorithm (see Class17 notes). It shows a sub-procedure that has the job of swapping the pivot row with the selected row below it. After that, it checks the new pivot element. If that number is too small, it stops, sensing that the set of equations cannot be solved.

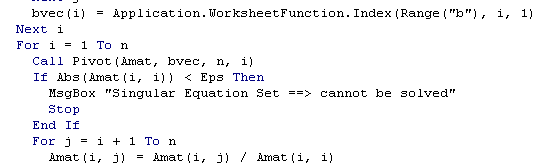
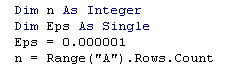


The pivot-swap procedure is then developed in the flowchart below.

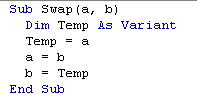
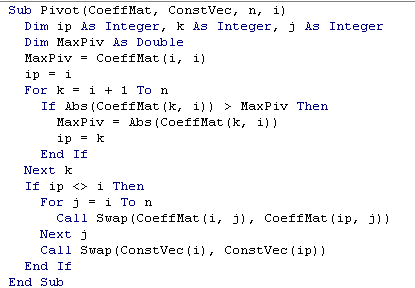


Now, you will modify the VBA code in the **Gauss.xlsm** spreadsheet to incorporate this partial pivoting strategy.

Modify the code in Sub Gauss as shown in the two code blocks below, adding the code in rectangles. Be careful to locate the new code correctly.



Then, at the end of Sub Gauss, after the End Sub statement, add the following two procedures:



Now, unless you have typographical errors or other bugs, you should be able to run the Gauss macro and get the correct solution to the equations.

Save your workbook as **GaussPivot.xlsm** and close it out.

3. Message Boxes

You have used simple message boxes in previous exercises and applications. Now, you will take a look at the MsgBox function in more detail.

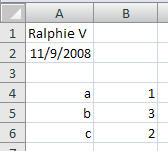
Open a new workbook, and put your name and date in cells A1 and A2. You will create a simple Excel/VBA application that solves the quadratic equation,



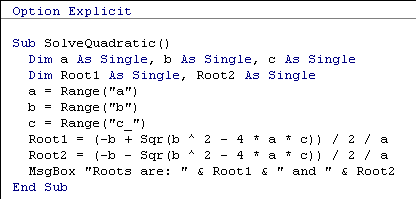
for given values of , , and . Enter labels **a**, **b** and **c** in cells A4:A6. With A4:B6 selected, using Formulas 🡺 Create from Selection, transfer the labels in cells A4:A6 as names on the cells B4:B6. Check the names that were created using the Name Box. Do you note anything unusual?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

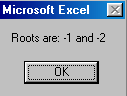
Put three trial values in cells B4:B6 as shown below.



Save your workbook as **Quadratic.xlsm**. Open the VBE and insert a module (make sure you are in the project associated with this workbook). Enter the following VBA procedure:



Run this Sub, and you should see



Fix any bugs until you get the program to run successfully. This illustrates the simple use of a message box.

This VBA program has limitations. Change the value of **b** on the spreadsheet to **1** and run it again. What do you observe?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Why does this occur?

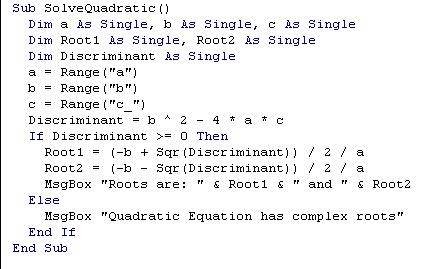
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

So, we now have to refine/improve the program to handle this situation. The key quantity that determines whether the calculation will encounter a problem is called the *discriminant* and is given by the formula



If this quantity is negative, the quadratic equation will have complex roots; otherwise, it will have real roots.

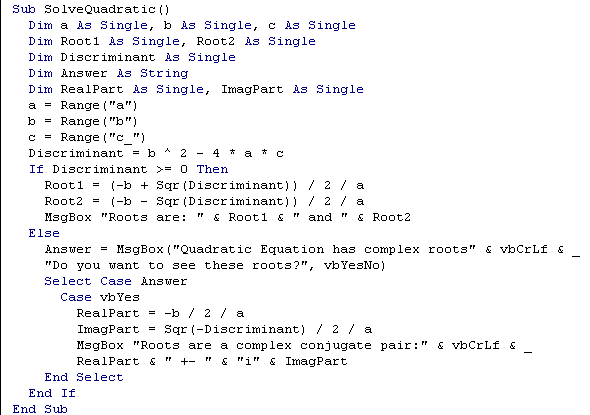
So, we modify the code to check for this and provide information on the situation. Make the modifications shown below:



Run this modified program and confirm that it "traps" the situation when the discriminant is < 0.

So, now our program is protected against failing when the coefficients yield an equation with complex roots. But this is not really satisfactory if we're interested in knowing what the complex roots are, at least occasionally.

We would like a design where the program would let the person using it know that the equation has complex roots and ask whether to proceed in displaying the complex roots or just quit. This can also be done with a message box, but the syntax is a bit more complicated. Make the following changes to your code to implement this.



Test this program with the coefficients that yield complex roots. Run it twice to check out the Yes and No buttons to make sure they work right.

There are still a couple ways that this program can get fouled up. Change the value in the **a** cell to zero and run the program.

What do you observe?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

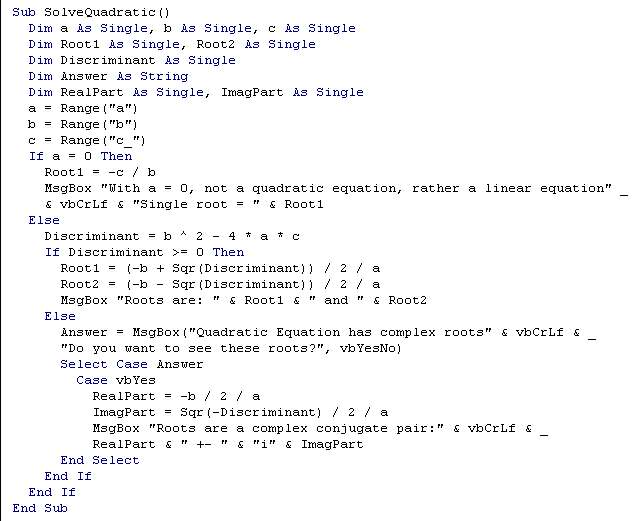
Why did this happen?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Let's protect against this problem too. Modify the SolveQuadratic program as shown below:

increase

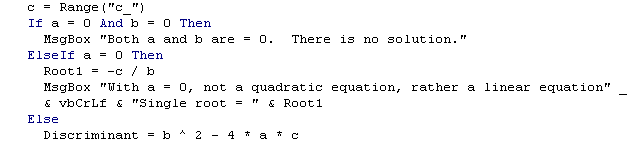
indent



Test this program with a zero in the a cell to make sure it "traps" that error properly.

We still have one remaining problem that could occur. What if both the a and b cells contain zeros. Try that out.

An additional modification is shown to the code to trap that error. Make the changes and test the program.



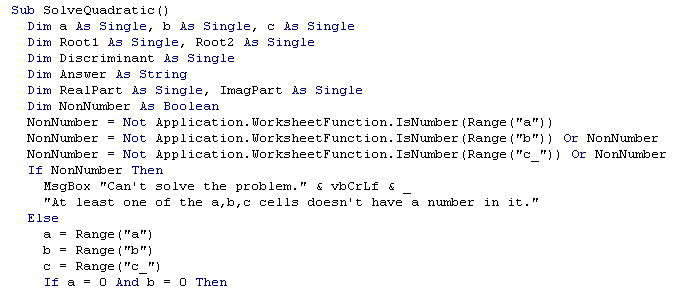
We have used a multi-alternative If structure (If/Then/ElseIf/Else) to handle the 3 cases:

* a and b = 0
* just a = 0
* neither a nor b = 0

It might also be possible, for whatever reason, that the **a**, **b** or **c** cells would be empty or contain text. We can check for that with a special function in Excel, ISNUMBER[[1]](#footnote-1). We now add another layer of protection to the program.

increase

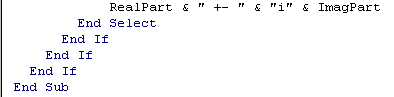
indent



•

•

•



Enter some text in one of the **a**,**b**,**c** cells and run the program to check that it traps this error. Then put numbers back in the cells to make sure it can still solve a normal problem.

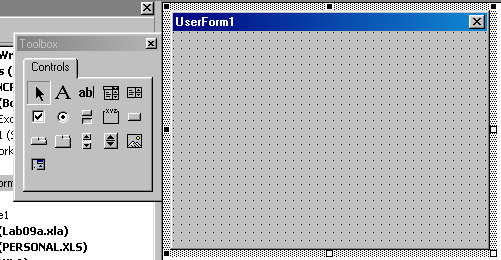
Save your workbook file and close it out.

4. User Forms (Custom Dialog Boxes)

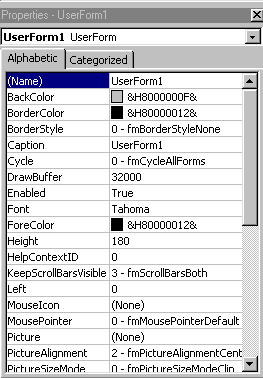
VBA provides a very powerful tool for custom user interfaces, the User Form. In this lab, you will get a start at creating User Forms, also called Custom Dialog Boxes. Open a new workbook and save it immediately as **TempConvert.xlsm**. Switch over to the VBE and make sure the correct VBA project is selected in the Project Explorer window. Use the Insert menu to create a UserForm.



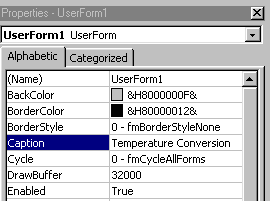
The VBE will change in appearance with a blank userform showing along with a Toolbox palette.



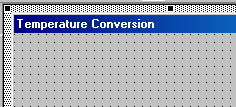
The Properties Window now becomes quite important.



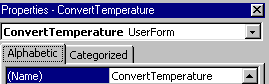
We will make changes to the UserForm by changing the entries in this window. Start by selecting the Caption entry with the mouse and change the caption to "Temperature Conversion", as shown in the figure below.



Notice that, immediately, the caption on the userform updates.



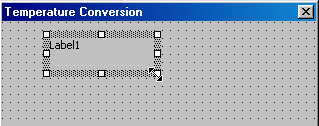
In the Properties Window, change the (Name) field to "ConvertTemperature". This renames the object that is the user form. The Properties Window should now look like



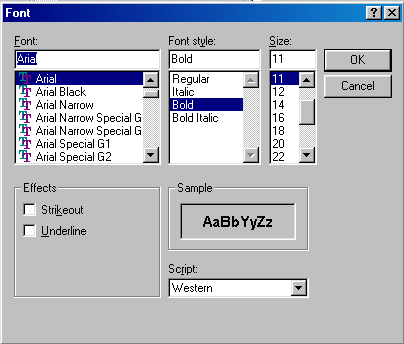
Now, select the Label tool from the Toolbox palette.



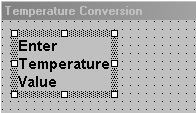
Click on the Label tool, then move the mouse to the userform and drag out a label rectangle that looks like



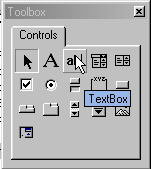
Notice that when this label object is selected the Properties Window displays its properties, not those of the userform. Click on the Font property and the button to the right of that field, , and set the font to Arial, the size to 11, and set the Font style to bold, as shown below.



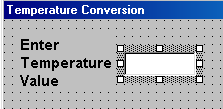
Then, click OK. You may need to resize the label box to look like



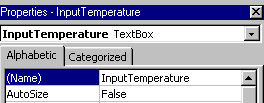
Now, click on the Text Box button in the Toolbox palette:



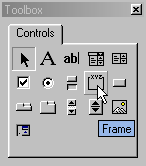
and drag out a text box in the userform to the right of the label to look like



With the text box selected, change the Name field in the Properties Window to "InputTemperature".

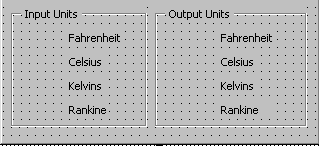


Select the Frame Tool and drag out an Input Units frame and an Output Units frame below the previously place label and text box.

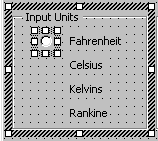
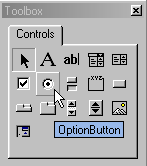


Use the Caption field in the Properties Window to change the frame captions as shown.

Now, using the label tool, place labels as shown below. You might need to stretch the frames some.



Select the Option Button Tool and place drag out an option button just to the left of the Fahrenheit label in the Input Units frame. Save your workbook for safety's sake.

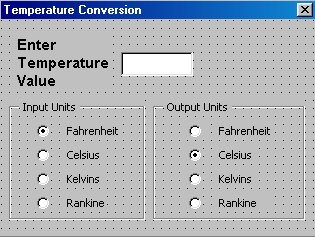


With this option button selected, use the Properties Window to change the Name field to

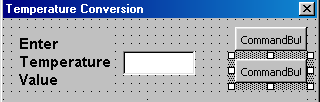
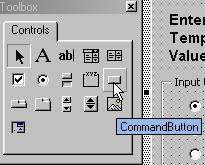
"FahrenheitInput" and change the Value field to True. The button should appear selected. It will be selected as the default input unit.

Add option buttons next to the rest of the unit labels and change their names to "CelsiusInput", . . ., "FahrenheitOutput", etc. Change the Value field for "CelsiusOutput" to True.

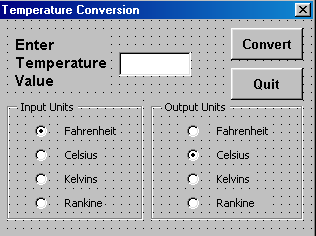
The userform should now look like



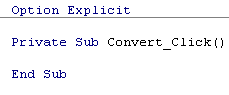
Now, we need to place a couple buttons that will allow us to make things happen. Use the CommandButton tool and place the buttons as shown.



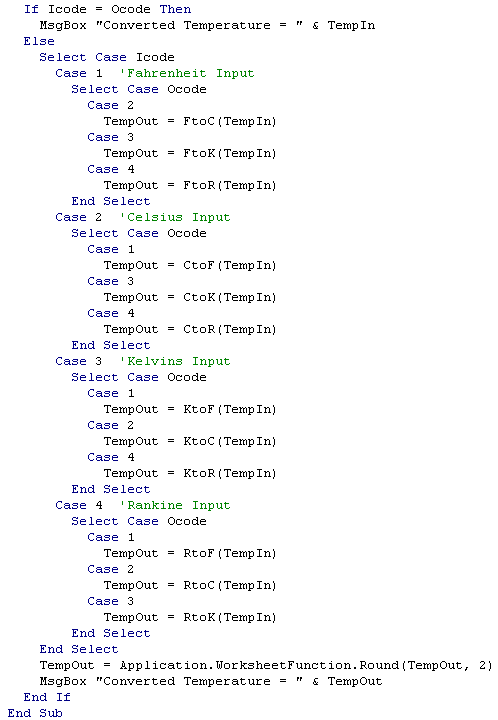
Select the upper button and change its Name field to "Convert" and its caption to "Convert". Select the lower button and change its Name field and caption Field to "Quit".



Now, we need to write the VBA code that executes when the Convert and Quit buttons are clicked. Start by double-clicking the Convert button. A code window should open up as shown below.



Save your workbook for safety's sake. Enter carefully the block of code shown below and on the next page between the Private Sub and End Sub statements.



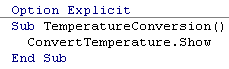
Select View → Object (Shift-F7 shortcut) and then double-click on the Quit button. You should see, appended to your previous code,



Enter the one statement shown below in this Sub:



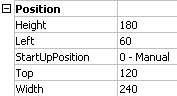
In order to test your program, select Insert → Module, and enter the short Sub:



Save your workbook for safety's sake. When you run this Sub, the user form should appear and you can test your program. You may want to do this by single-stepping this Sub to follow the progress of the program.

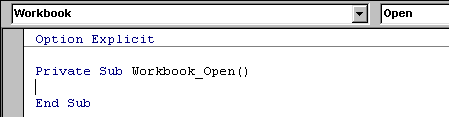
When you are testing the program, you will notice that the message box result sits right on top of the user form, making it impossible to see the units choices that were made. To remedy this, we can position the user form off to the left.

Select the user form again from the Project Explorer and change its properties as shown below. In doing so, click on the Categorized tab in the Properties Window and bring the Position category into view. Make three changes so that category looks like

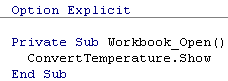


Test your program again and see that the user form and the message box can be seen separately.

Finally, you'll add an "event handler" that brings up your user form when the workbook is opened. In the VBE, double-click on the ThisWorkbook item in the Project Exporer. Change the left field at the top of the code window to Workbook (use the drop-down list). You should now see



The automatically-created Workbook\_Open Sub allows you to enter code that will be executed when the "event" of the workbook being opened occurs. Enter the one statement that will display the user form, as shown below



Now, save your workbook and close it out. Re-open your workbook, and the user form should appear.

Use your Excel/VBA application to complete the table below:

|  |  |  |
| --- | --- | --- |
| 37 °C = \_\_\_\_\_\_\_\_ °F | 212 °F = \_\_\_\_\_\_\_\_ °C | **-** 40 °C = \_\_\_\_\_\_\_\_ °F |
| 100 K = \_\_\_\_\_\_\_\_ °C | 1000 °R = \_\_\_\_\_\_\_\_ °F | 25°C = \_\_\_\_\_\_\_\_ K |

5. Leave Excel and return to Windows.

**End of Lab Workshop #9**

1. Since it is an Excel function, we'll have to refer to it with Application.WorksheetFunction.ISNUMBER [↑](#footnote-ref-1)