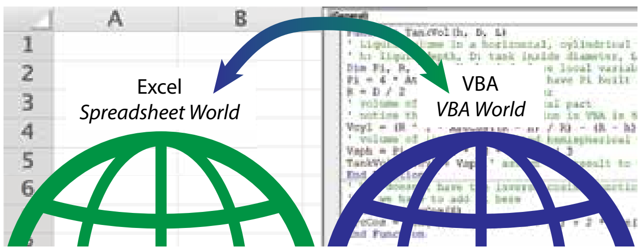
Improve Your Spreadsheet Productivity with VBA

David E. Clough
Univ. of Colorado

Excel includes a programming language that can improve your productivity, as well as the reliability of your spreadsheets. Simple programs in Visual Basic for Applications (VBA) can have a big impact on your work for little effort.

Most Excel users, including chemical engineers, have not ventured behind the curtain to leverage the Visual Basic for Applications (VBA) programming language that accompanies the spreadsheet program. Several reasons for this include: many chemical engineers have picked up the use of Excel on the fly; the availability and utility of VBA is not obvious, and its use is not required; and many chemical engineers have a disdain for programming languages, often instilled by an unpleasant experience in a computer programming course during college.

Because integrating VBA into spreadsheet problem-solving can bring many benefits in terms of efficiency, power, and reliability, the purpose of this article is to break the ice. Examples of a few elementary applications of VBA are presented. Using similar concepts in your spreadsheets can help make you more productive for little effort. This article builds on the concepts presented in the August 2016 *CEP* article "Use Spreadsheets for ChE Problem-Solving."



▲ **Figure 1.** The Visual Basic Editor opens as a separate window from your spreadsheet, making the two environments seem like different worlds that pass information back and forth.

Evolution of spreadsheet programming

Spreadsheet software was developed on personal computers, such as VisiCalc on the Apple IIe and Lotus 1-2-3 on the IBM PC, well before the advent of the graphical user interface and the mouse pointing device. The menus in the spreadsheet programs were operated using keystrokes. For example, in Lotus 1-2-3, the / key activated the menu, the arrow keys navigated the menu options, and the Enter key made a selection (pressing the first letter of a menu option, *e.g.,* D for Data, was a quicker way to also make a selection). Submenus could be operated in the same way. We are now accustomed to using the mouse to make menu selections, although the keyboard (using the Alt key) can still be used to navigate the Excel interface. If memorized, the key sequence can be much faster than using the mouse.

Early in spreadsheet program development, the programs incorporated a "macro" feature that allowed the user to enter a series of keystrokes in a cell or a series of contiguous cells and identify it with a shortcut key combination, which enhanced efficiency for common menu sequences. Numerous competitors entered the spreadsheet market in the late 1980s and early '90s with macro language features that enabled the use of a limited programming structure, such as decisional branches and count-controlled loops. These macro languages evolved to a fairly complex level (*e.g.,* Excel's Macro Language Version 4.0) but were never designed from the ground up as programming languages. They also allowed the recording of macro programs by following, and later mimicking, the sequence of keystrokes that drive the menus.

Microsoft overhauled Excel's programming capabilities in 1993 with the introduction of VBA in Excel Version 5.0. VBA is an object-oriented programming language that is fully featured and is used across the Microsoft Office platform, including versions in Access, Word, and PowerPoint. With the launch of VBA, the programming capabilities became less integrated with Excel and, in essence, live in their own "world" (Figure 1).

A fully featured, somewhat independent, programming language like VBA has major benefits but also has some drawbacks. When recording macros for a series of tasks, VBA observes what the user is doing on the spreadsheet and interprets it as a VBA program. Often, the interpretation is not exactly what the user intended and must be fixed. There are also difficulties associated with the exchange of information (*e.g.,* data) between the spreadsheet and VBA environments. And, hundreds of useful built-in functions in Excel are not directly available in VBA, which has a limited set of functions.

Despite limitations, chemical engineers can use VBA to:

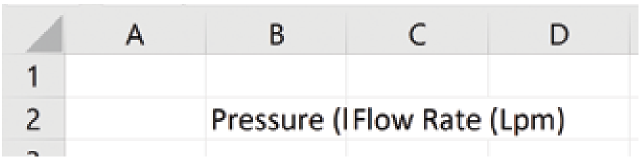
• *record macros.* Macros that automate repetitive tasks offer the greatest opportunity to improve your efficiency when using spreadsheets.

• *develop custom functions.* Custom user-defined functions, especially for commonly used, formula-based engineering calculations, help streamline spreadsheet operation.

• *create VBA programs*. Stand-alone VBA programs that carry out more-complex calculations involving information contained in the spreadsheet are powerful tools.

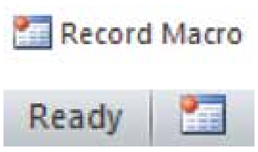
• *make custom interfaces.* Custom user applications may include user forms and interfaces with other software, such as data acquisition tools and process simulators.

This article focuses on macros and custom functions. Chemical engineers who use Excel for day-to-day problem-solving can benefit greatly by leveraging these two simple tools. The latter two functions are introduced in this article but require more in-depth coverage. (Two courses in the AIChE Academy, available in both online and in-person formats, address creating VBA programs and custom interfaces.)

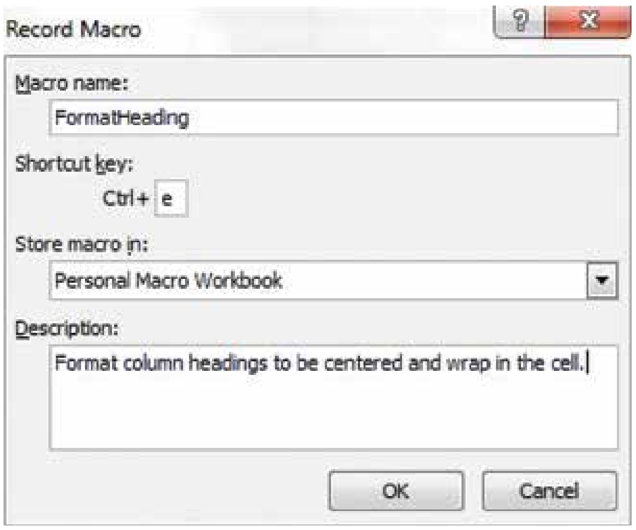


	A	B	C	D
1				
2		Pressure (I	Flow Rate (Lpm)	

▲ **Figure 2.** The complete headings in cells B2 and C2 — Pressure (kPa) and Flow Rate (Lpm) — are longer than the widths of columns B and C.



◀ **Figure 3.** After you select the Record Macro command button in the Code group of the Developer tab of the Ribbon or the Record Macro icon on the left of the status bar below the spreadsheet, VBA will record your actions in the spreadsheet environment.



Record Macro

Macro name: FormatHeading

Shortcut key: Ctrl+ e

Store macro in: Personal Macro Workbook

Description: Format column headings to be centered and wrap in the cell.

▲ **Figure 4.** Be careful when assigning shortcut keys to macros because they will supersede any built-in shortcut key definitions. Most uppercase shortcut keys are available, but avoid F, O, and P. Many lowercase shortcut keys are already defined, although the e, j, l, m, q, and t keys are generally available. Third-party add-in software might occupy other shortcut keys.

The Excel-VBA interface: Moving between the worlds

VBA programs are developed, edited, and managed in the Visual Basic Editor (VBE) environment. There are numerous ways to move back and forth between the Excel spreadsheet and the VBE. (This article will cover the methods suitable for working with Microsoft Office 2016, and the associated version of Excel, for the Microsoft Windows 10 operating system. Earlier versions on Windows-based systems do not vary too much.) Keying Alt-F11 is the easiest way to open the VBE. Alternatively, you can select the Visual Basic icon from the Code group on the Developer tab of the Ribbon. (If the Developer tab does not appear on your Ribbon, activate it via File, Options, Customize Ribbon, and check the box next to Developer.) Once the VBE is activated, you can switch back to Excel by keying Alt-F11, by clicking the Excel icon on the left side of the toolbar, or by selecting the Excel icon on the Windows taskbar.

The VBE layout harkens back to pre-Ribbon versions of Office, which had a menu and toolbar at the top, a Project Explorer window on the upper left, a Properties window on the lower left, and the main area called the Code Window.

Record VBA macros: A big bang for a pocket-change investment

As we use spreadsheets in our daily chemical engineering work, we inevitably come across multistep procedures that we repeat again and again. In many cases, we can record a macro to automate these simple but repetitive procedures. Each of us will have our own examples, but here are four of mine:

- centering and wrapping column headings that are wider than the numerical entries below them
- right-justifying a cell entry one space from the border
- transferring a label as the name on the adjacent cell to the right
- resetting the zoom setting of new, blank worksheets to 150%.

Consider the first repetitive task: I would like to center and wrap two adjacent headings that exceed their column width (Figure 2). To do this, select the two cells and then activate the Macro Recorder — either by clicking the command icon in the Code group of the Developer tab of the Ribbon or by clicking the Record Macro icon on the left of the status bar below the spreadsheet (Figure 3). Once activated, the Record Macro dialog box is superimposed over the spreadsheet.

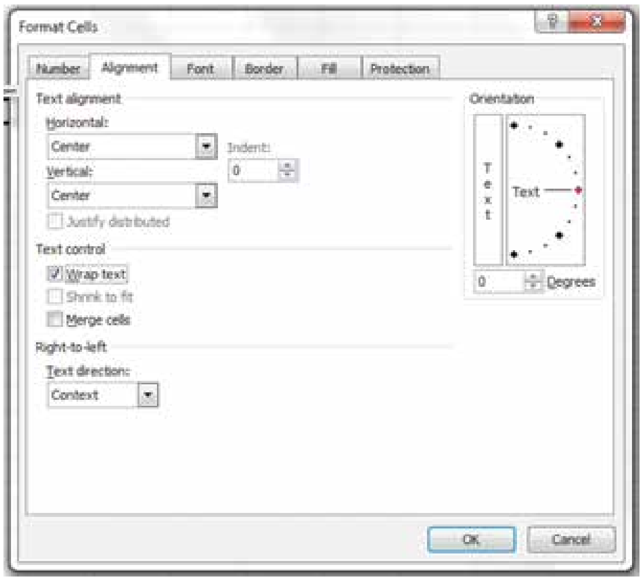
Enter into the Record Macro dialog box a name for the macro and a shortcut key (Figure 4). Storing the macro in the Personal Macro Workbook makes it available to whatever Excel workbook is open. If you have not yet stored anything in the Personal Macro Workbook, it will be created for you when you record your first macro to be stored there. Finally, you should enter a few words in the Description field related to the function of the macro.

Click OK to turn on the recorder. From this point on, VBA observes what you do in the spreadsheet environment and attempts to compose code to replicate it. To center and wrap the text in the cells of interest, right-click the selected cells and select Format Cells. Navigate to the Alignment tab and in the dialog box select Center from the Horizontal and Vertical drop-down menus and 0 from the Indent menu, and check the Wrap text box, as shown in Figure 5. Click OK to format the cells as shown in Figure 6. To stop recording, select the Stop Recording command icon on the Ribbon or the Stop Recording square on the Status Bar, which is next to the Ready status (Figure 7).

Test the macro by typing another lengthy label, selecting the cell containing the label, and keying the shortcut combination. If the macro does not work, examine the VBA code. In fact, it is not a bad idea to look at the code anyway to see an example of the code or check for errors.

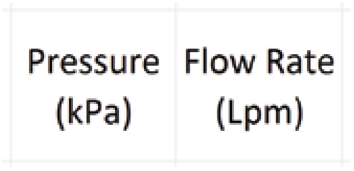
Key Alt-F11 to switch to the VBE. Double-click the code module in the Project Explorer under the Project associated with the Personal Macro workbook (PERSONAL.XLSB). The Code Window will open and should be populated with code (Figure 8).

If there are errors, they will probably show up as extra lines. The Sub (*i.e.,* Subroutine Procedure) presented in Figure 8 is evidence of the relative readability of VBA program code. Comments are in green and set off by a leading apostrophe. The nine statements inside the With Selection ... End With structure refer to the options on the



Text alignment
Horizontal: Center
Indent: 0
Vertical: Center
☐ Justify distributed
Text control
☑ Wrap text
☐ Shrink to fit
☐ Merge cells
Right-to-left
Text direction: Context
Orientation
0 Degrees

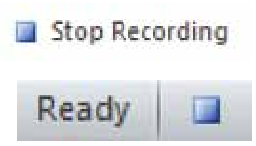
▲ **Figure 5.** Under the Alignment tab of the Format Cell window, specify Center for both Horizontal and Vertical text alignment and check the Wrap text box.

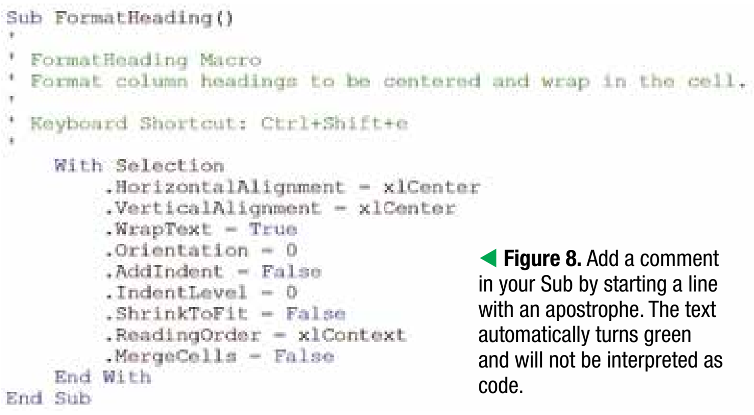


Pressure (kPa)	Flow Rate (Lpm)

◀ **Figure 6.** The selections made in the Format Cell window format the column headings so they are easier to read.

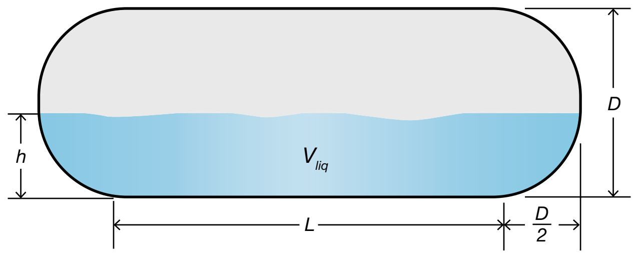
▶ **Figure 7.** When you are ready to stop recording your macro, select the Stop Recording command icon on the Ribbon or the Stop Recording square on the Status Bar (next to the Ready status).



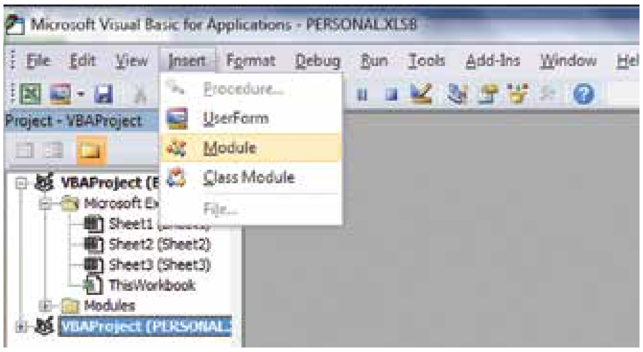


```
Sub FormatHeading()
'
' FormatHeading Macro
' Format column headings to be centered and wrap in the cell.
'
' Keyboard Shortcut: Ctrl+Shift+e
'
    With Selection
        .HorizontalAlignment = xlCenter
        .VerticalAlignment = xlCenter
        .WrapText = True
        .Orientation = 0
        .AddIndent = False
        .IndentLevel = 0
        .ShrinkToFit = False
        .ReadingOrder = xlContext
        .MergeCells = False
    End With
End Sub
```

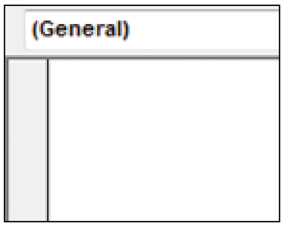
◀ **Figure 8.** Add a comment in your Sub by starting a line with an apostrophe. The text automatically turns green and will not be interpreted as code.



▲ **Figure 9.** Create a user-defined function (UDF) to calculate the volume of liquid in this horizontal cylindrical tank with hemispherical sides.



▲ **Figure 10.** After you select your project in the Project Explorer window, select Insert and Module on the VBE menu to open a blank Code Window.



◀ **Figure 11.** The blank code window in the VBE is where you enter your code.

▼ **Figure 12.** Comments can help make your UDF more easily understood by others, and they can help remind you of your reasoning when you revisit the code.

Alignment tab of the Format Cells dialog window. Selection refers to the cells that are selected, which identifies the object to which the code applies. Each item following a period is a property associated with that object, which is assigned a value via the equals sign.

Coding the word Selection in front of each period of the nine lines, absent the With Selection … End With, accomplishes the same thing as the With Selection ... End With statement; however, the latter is more concise.

Only the first three lines of the macro correspond to the selections made on the Alignment tab. The last six are default values that can be deleted to make the code more compact and easy to understand.

This short time investment produces a simple macro that will likely be used dozens, perhaps hundreds, of times. Similar short macros can just as easily be recorded and provide a big payback.

User-defined VBA functions: Build your arsenal of engineering formulas

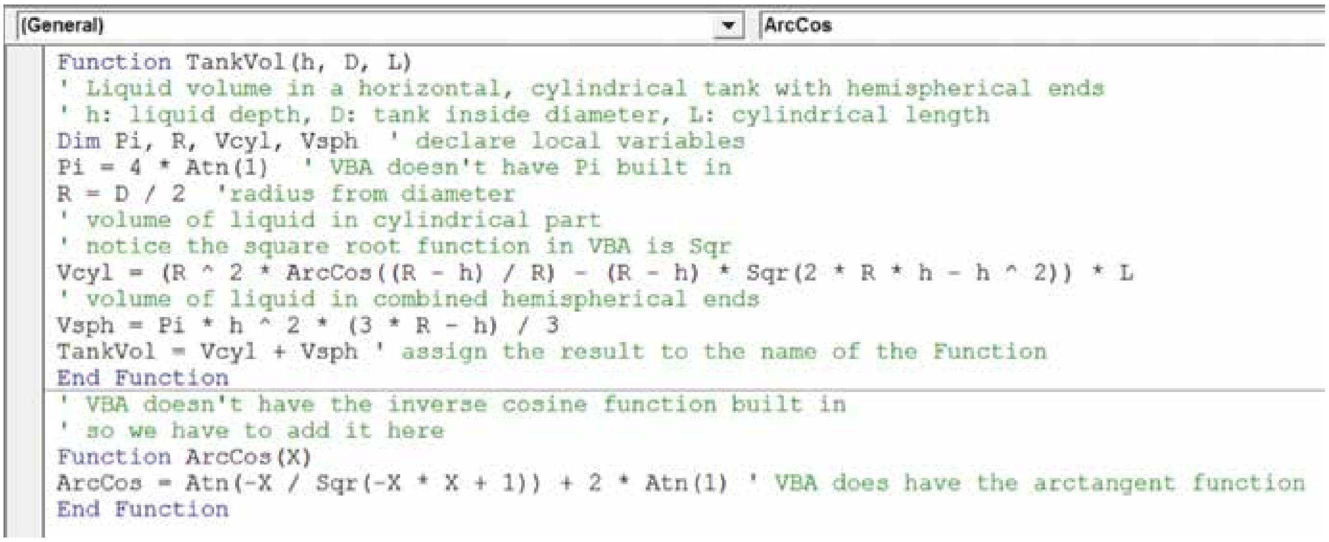
Chemical engineers occasionally use formulas for calculations that are relatively complex. These formulas can be incorporated into a user-defined VBA function (UDF). A UDF:

• allows the formula to be invoked in numerous locations on the spreadsheet without you re-entering or copying it

• improves reliability because it ensures only one version of the formula is used in the spreadsheet

• provides the ability to package the UDF into an add-in, possibly with other UDFs, and share the add-in with a work group or company organization (even commercialize it) to improve consistency of application.

To see how formulas can be incorporated into UDFs, consider the volume of liquid in a horizontal, cylindrical



```
Function TankVol(h, D, L)
' Liquid volume in a horizontal, cylindrical tank with hemispherical ends
' h: liquid depth, D: tank inside diameter, L: cylindrical length
Dim Pi, R, Vcyl, Vsph  ' declare local variables
Pi = 4 * Atn(1)   ' VBA doesn't have Pi built in
R = D / 2  'radius from diameter
' volume of liquid in cylindrical part
' notice the square root function in VBA is Sqr
Vcyl = (R ^ 2 * ArcCos((R - h) / R) - (R - h) * Sqr(2 * R * h - h ^ 2)) * L
' volume of liquid in combined hemispherical ends
Vsph = Pi * h ^ 2 * (3 * R - h) / 3
TankVol = Vcyl + Vsph ' assign the result to the name of the Function
End Function
' VBA doesn't have the inverse cosine function built in
' so we have to add it here
Function ArcCos(X)
ArcCos = Atn(-X / Sqr(-X * X + 1)) + 2 * Atn(1) ' VBA does have the arctangent function
End Function
```

tank with hemispherical ends (Figure 9). The total volume of liquid in the tank (V_{liq}) can be calculated by summing the volume of liquid in the cylindrical portion of the tank (V_{cyl}) and the volume of liquid in the hemispherical ends (V_{sph}) as defined by:

$$R = \frac{D}{2} \tag{1}$$

$$V_{cyl} = L\left[R^2 \cos^{-1}\left(\frac{R-h}{R}\right) - \left(R-h\right) - \sqrt{2Rh-h^2}\right] \tag{2}$$

$$V_{sph} = \frac{\pi h^2 \left(3R-h\right)}{3} \tag{3}$$

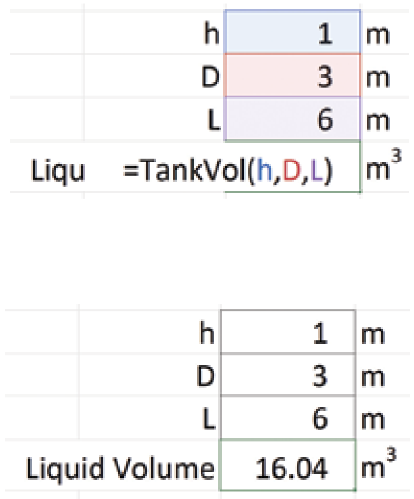
$$V_{liq} = V_{cyl} + V_{sph} \tag{4}$$

where R is the radius of the tank, D is the diameter of the tank, h is the depth of the liquid in the tank, and L is the length of the cylindrical portion of the tank.

To create a UDF with the format =TankVol(h, D, L), enter code directly into the VBE. Select the appropriate project corresponding to the Excel Workbook in the Project Explorer window, and then select Insert and Module on the VBE menu (Figure 10) to open a blank Code Window (Figure 11).

Figure 12 shows the Code Window populated with the appropriate code. A good practice is to include ample comments to describe the calculation. The next step is to test the function on the spreadsheet (Figure 13).

Other formulas can be implemented in UDFs but might require program structure, such as decisions (If statements) and iterations (For-Next and Do-Loop statements). Books that cover the details of writing VBA code are available; a good one is *Excel 2016 Power Programming with VBA* by M. Alexander and D. Kusleika (John Wiley & Sons, Inc., Feb. 2016).



h	1	m
D	3	m
L	6	m
Liqu	=TankVol(h,D,L)	m^3

h	1	m
D	3	m
L	6	m
Liquid Volume	16.04	m^3

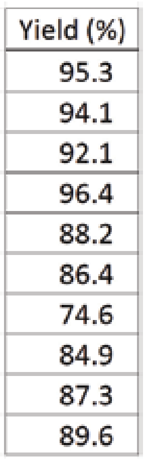
▶ **Figure 13.** The variables *h*, *D*, and *L* are the necessary input for the function =TankVol(). In this spreadsheet, the cells have been renamed h, D, and L, rather than retaining their original coordinate names (*i.e.,* B2, B3, B4).

Streamline and format data: Save time and improve reliability

Data sets often need to be cleaned up and formatted to improve their readability and usefulness. If you find yourself doing this repeatedly, consider developing a VBA program to streamline the effort.

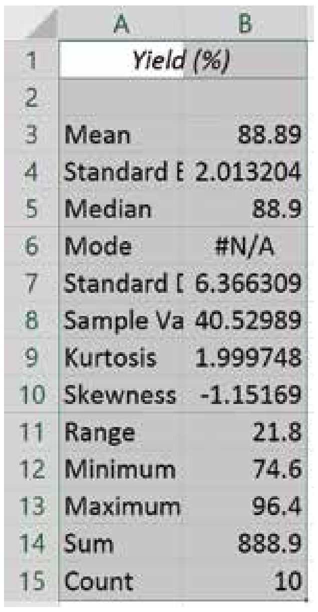
A small-scale example of data in need of formatting is the output of Excel's Analysis Toolpak feature. The Analysis Toolpak has a Descriptive Statistics option that generates a statistical analysis of a data set. If you use the Toolpak option for Descriptive Statistics for a small set of data (Figure 14) and designate its output for a separate, new worksheet, the Toolpak provides a table summarizing an analysis of the data set such as in Figure 15.

To make this table more readable manually, we would adjust column widths and number formats, and we could record a macro to see the code composed by VBA for these operations (Figure 16). I cleaned up the code in Figure 16 by removing superfluous statements, and I specified a particular numeric format for each cell. This would be a limitation for a generic macro but would be sufficient if the data sets we treat are consistent. Adding code to display a certain number of significant figures, independent of the scale of



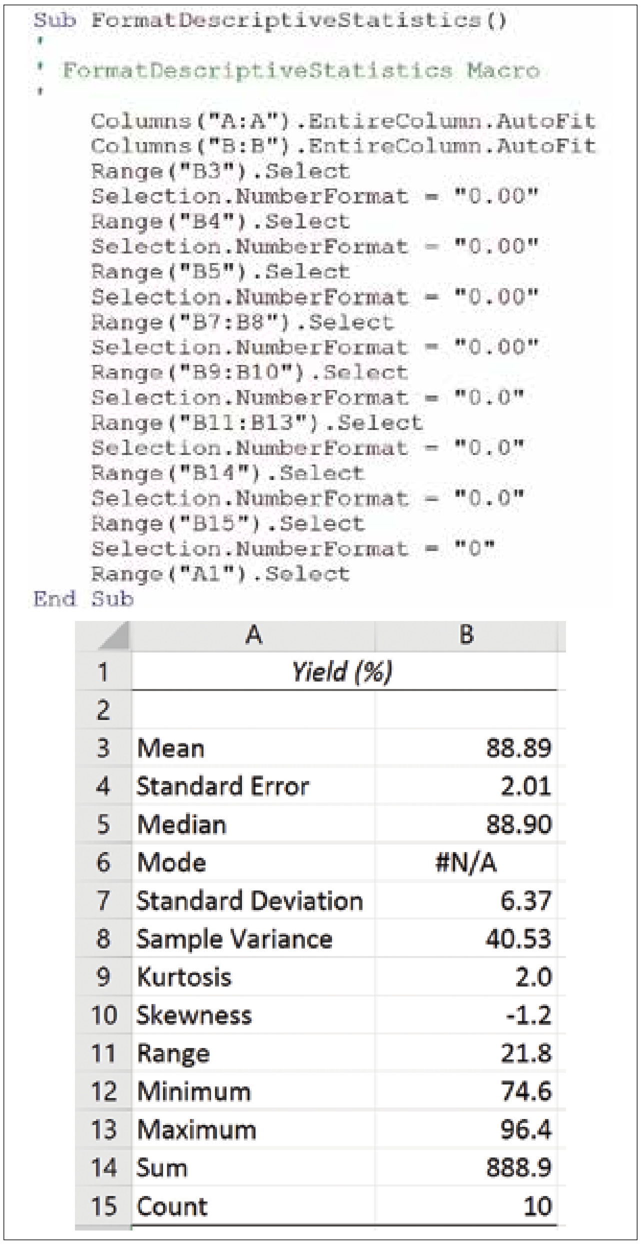
Yield (%)
95.3
94.1
92.1
96.4
88.2
86.4
74.6
84.9
87.3
89.6

◀ **Figure 14.** We could analyze this set of data manually. However, Excel has an automated analysis feature that is particularly useful for large sets of data.



	A	B
1	Yield (%)	
2		
3	Mean	88.89
4	Standard E	2.013204
5	Median	88.9
6	Mode	#N/A
7	Standard D	6.366309
8	Sample Va	40.52989
9	Kurtosis	1.999748
10	Skewness	-1.15169
11	Range	21.8
12	Minimum	74.6
13	Maximum	96.4
14	Sum	888.9
15	Count	10

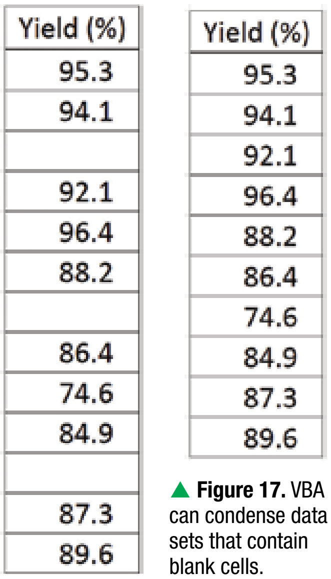
▶ **Figure 15.** Excel's Analysis Toolpak includes a Descriptive Statistics option, which provides statistics about data sets in the form of a table.



```
Sub FormatDescriptiveStatistics()
'
' FormatDescriptiveStatistics Macro
'
    Columns("A:A").EntireColumn.AutoFit
    Columns("B:B").EntireColumn.AutoFit
    Range("B3").Select
    Selection.NumberFormat = "0.00"
    Range("B4").Select
    Selection.NumberFormat = "0.00"
    Range("B5").Select
    Selection.NumberFormat = "0.00"
    Range("B7:B8").Select
    Selection.NumberFormat = "0.00"
    Range("B9:B10").Select
    Selection.NumberFormat = "0.0"
    Range("B11:B13").Select
    Selection.NumberFormat = "0.0"
    Range("B14").Select
    Selection.NumberFormat = "0.0"
    Range("B15").Select
    Selection.NumberFormat = "0"
    Range("A1").Select
End Sub
```

	A	B
1	*Yield (%)*	
2		
3	Mean	88.89
4	Standard Error	2.01
5	Median	88.90
6	Mode	#N/A
7	Standard Deviation	6.37
8	Sample Variance	40.53
9	Kurtosis	2.0
10	Skewness	-1.2
11	Range	21.8
12	Minimum	74.6
13	Maximum	96.4
14	Sum	888.9
15	Count	10

◀ **Figure 16.** The number of decimal places for each cell in the range can be individually specified in the code (top).



Yield (%)
95.3
94.1
92.1
96.4
88.2
86.4
74.6
84.9
87.3
89.6

Yield (%)
95.3
94.1
92.1
96.4
88.2
86.4
74.6
84.9
87.3
89.6

▲ **Figure 17.** VBA can condense data sets that contain blank cells.

the numbers, requires more effort.

Another elementary application of VBA is to remove blank cells from a set of data. Compressing a data set with a few blank cells (Figure 17) requires a VBA Sub with a little program structure (Figure 18).

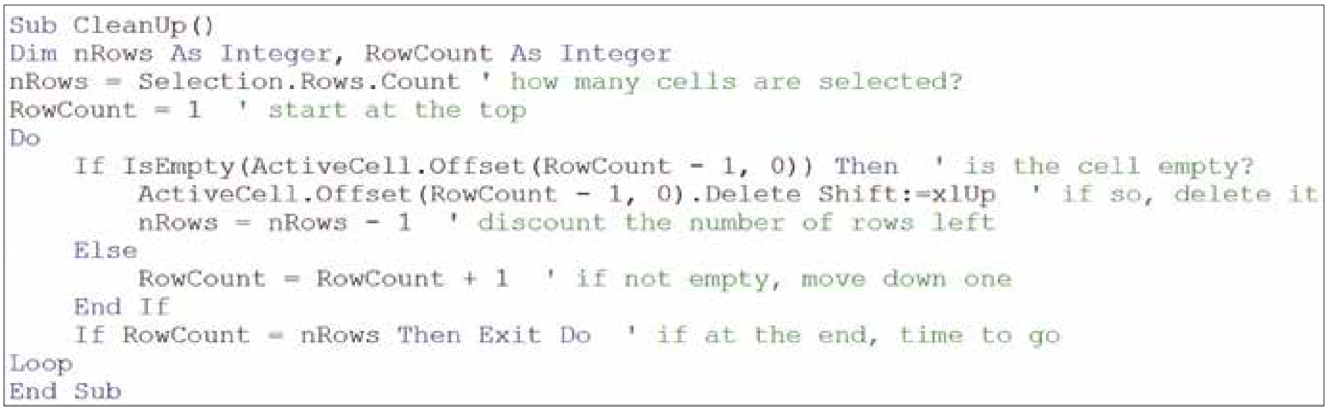
User interfaces: Applications to be used by others

If you would like to develop a spreadsheet to be used by others, such as operators and technicians, consider creating a user interface as part of your application. An efficient and well-designed spreadsheet interface for a common calculation that must be performed by those who are not spreadsheet-savvy can be worth the effort.

A user interface (Figure 19) requires the design of objects, such as forms, message boxes, input boxes, buttons, and switches, which are generally linked to VBA event handler code segments. Designing and programming these applications takes time and the learning curve is steeper than, for example, recording a macro.

Spreadsheets can also be integrated with other software packages, including math software (*e.g,* Matlab), process simulators (*e.g.,* Aspen Plus), optimizers (*e.g.*, What'sBest!),

▼ **Figure 18.** You can discover statements like .Delete Shift:=xlUp by recording a short macro to delete a cell. You can use this information to then write a custom Sub.



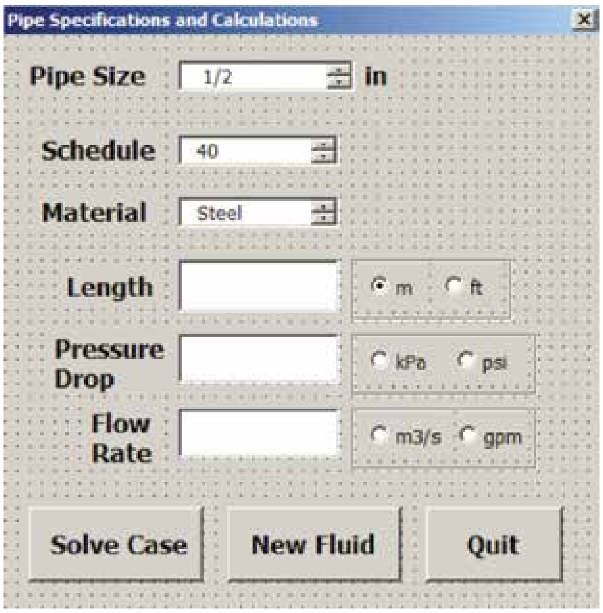
```
Sub CleanUp()
Dim nRows As Integer, RowCount As Integer
nRows = Selection.Rows.Count ' how many cells are selected?
RowCount = 1  ' start at the top
Do
    If IsEmpty(ActiveCell.Offset(RowCount - 1, 0)) Then  ' is the cell empty?
        ActiveCell.Offset(RowCount - 1, 0).Delete Shift:=xlUp  ' if so, delete it
        nRows = nRows - 1  ' discount the number of rows left
    Else
        RowCount = RowCount + 1  ' if not empty, move down one
    End If
    If RowCount = nRows Then Exit Do  ' if at the end, time to go
Loop
End Sub
```

and data acquisition software (*e.g.,* LabView, distributed control systems [DCS]). It is also possible to link spreadsheets with programs written in other programming languages, such as C/C++ or Fortran. In most cases, the bridge between Excel and these other packages is VBA.

Some software vendors provide Excel-based add-ins that facilitate intercommunication, such as the Excel Link add-in for Matlab and the Pi software from OSI. However, for the most flexibility and power, it is frequently necessary to write object-oriented VBA code.

Concluding thoughts

Most chemical engineers will not be able to carve out the time to get heavily involved in VBA application development. However, it is well within the reach of practicing professionals to take advantage of the small-scale features of VBA, such as macros and user-defined functions, and reap substantial benefits in time-savings and spreadsheet reliability. If you haven't tried these features, venture into the VBA world to see what it can offer. CEP



▲ **Figure 19.** Selecting the Solve Case button produces the desired output, which can appear in a Message Box or be placed on a Worksheet.

DAVID E. CLOUGH, PhD, has been on the faculty of the Dept. of Chemical and Biological Engineering at the Univ. of Colorado Boulder since 1975 (Email: david.clough@colorado.edu). He conducts research in the automatic control of a variety of processes, most recently of solar-thermal reactors. At the university, he teaches courses in instrumentation and control, applied statistics, and engineering computing. Since 1989, he has taught well over 100 offerings of short courses in spreadsheet problem-solving and VBA programming for AIChE. His short courses have been among the most popular offered by AIChE and continue in both in-person and online format. For more than two decades, his co-instructor was Miles Julian of the DuPont Co., who is now retired.