INTRODUCTION TO TKSOLVER

By Robert L. Norton P. E.

INTRODUCTION TO TKSOLVER

The *TKSolver* program possesses some unique features that are very useful for solution of design problems in general and for machine-design problems in particular. These features include the ability to backsolve for any variable by simply switching it from an input column to an output column and then moving some other variable from output to input. In addition the program will automatically invoke a root-finding algorithm applying user-supplied guess values for the unknown parameters. It can thus solve equations in which the unknown parameters are implicit (i.e., appear more than once in the equation). Systems of simultaneous nonlinear equations can also be solved by its root-finding algorithm. It comes with an extensive library of prewritten functions that do many mathematical tasks such as numerical integration, numerical solution of differential equations, etc.

Design problems typically contain more variables than we have equations for and thus cannot be directly solved. Many assumptions (read intelligent guesses) must then be made to obtain a trial solution from which some insight is gained into the problem. The assumed values are then changed and the model is recalculated to obtain a better solution. *TKSolver* allows this iteration process to proceed in an easy and rapid manner.

TKSolver is a member of the class of programs known as **equation solvers** which typically allow the typing in of equations in a line-by-line fashion as you would do with any programming language. Unlike programming languages and most other equation solvers, which require the equation to be put in explicit form (i.e., the one unknown variable isolated on the left of the equal sign and all other terms on the right side), *TK* allows the equation to be input in **free form**. This may seem like a minor convenience but is really quite powerful, as it allows implicit variables to be solved as described above.

In fact, there is really no distinction between input and output variables within the equations of a *TK* model. Any variable can take on either input or output characteristics at different times. This is what allows “instant” back-solving. This means that you can, for example, specify the safety factor as an input variable, assign a desired value, and solve for any one of the part dimensions that affect the safety factor, no matter how complicated the equations or how many times that dimension’s variable appears in the collection of equations that define the engineering model. Once the model is calculated, output of the data in table or graph form is simple to accomplish.

There are many more useful features of this program. Rather than list them, it will be more productive to present a few simple examples to show what it can do. The *Student Manual* included with the software provides additional instruction for, and examples of, its use. Context-sensitive, on-line help is built into the program as well.

We will create a simple design problem to use as an example and gradually increase its complexity in succeeding examples as additional topics and features of the program are introduced. Before presenting the examples, we must define some of the terminology associated with this program.

Terminology

The *TKSolver* program contains a number of "sheets" which can be thought of as a paradigm for the sheets of paper that clutter the typical engineer's desk. One of these sheets, the **rule sheet,** contains the equations or "rules" that define the model. Another sheet, the **variable sheet,** contains the values of the known input parameters, and, after the rules are solved will also contain the output values. One of the advantages of *TKSolver* is its clean separation of rules (equations) and variables (data). The rules can (and should) be kept in strictly symbolic terms and the data changed on the variable sheet to recalculate or iterate the problem.

Beneath the variable sheet (and other sheets) are additional "subsheets" that contain more detailed information on each variable. These subsheets can be accessed from a pull-down menu or with a keystroke combination which differs depending on the version of *TKSolver* used.* This is referred to as "**diving**" down to the subsheet, as in rummaging beneath the papers on your desk to find that sheet with additional data on it. When the program is first run, both the rule and variable sheets are open and visible on the screen. Other sheets (**list**, **function**, **plot**, **table**, **unit**, **format**, and **comment**) are accessible from the WINDOW pull-down menu. The use of these sheets will be introduced in later examples.

Simple Models, Rule and Variable Sheets, Subsheets

We wish to design a 1-liter-capacity cooking pot, as shown in Figure 1, to be made of 1-mm-thick stainless steel. Ultimately we would like to minimize the amount of stainless steel used. We will start with the simplest possible model.

EXAMPLE 1

Simple Equation-Solving Using TKSolver - Part A

Problem	**Find the height and empty weight of an open-top, cylindrical container (a cooking pot) for a desired volume.**
Given	**The volume = 1 liter, inside diameter = 12 cm, wall thickness = 0.1 cm.**
Assumptions	**The material is stainless steel with a mass density of 7.75 g/cc.**
Solution	**See Figure 1, Tables 1 and 2, and the *TKSolver* file EX-01.tkw.**

1 An equation for the volume of a thin-walled cylinder can be expressed as

$$vol = basearea \cdot H \qquad (a)$$

where H is the height of the cylinder.

* In the *Windows* version, a "dive" to a subsheet is done with *Alt* + *Enter* or by clicking the right mouse button with the cursor on the item to be dived into. In the *DOS* version, use *Shift* + > to dive down and *Shift* + < to dive back up. In the *Macintosh* use *Command* + *mouse-click*. The *Windows* and *Macintosh* versions also provide pull-down menu picks for opening subsheets. These examples will refer to the *Windows* versions of these (and other) commands. See your manual for equivalents in other versions.

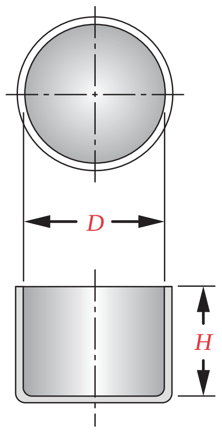

FIGURE 1

A Stainless Steel Cylindrical Container (Cooking Pot)

2 The area of the base can be found from

$$basearea = \frac{\pi D^2}{4} \tag{b}$$

where D is the inside diameter of the base.

3 Since we want to know the weight of the cooking pot, we need its total surface area, which can be found from

$$surface = basearea + \pi DH \tag{c}$$

4 The weight is then

$$surface \cdot thick \cdot dens = weight \tag{d}$$

5 These equations are encoded into the rule sheet as shown in Table 1. Though this model is provided on disk, it will be of more value if you create this model (and those in later examples) from scratch as described here. If you do that, you will notice that as soon as you hit ENTER after typing each equation, the variable names in that equation automatically appear on the variable sheet as shown in Table 2.

6 To solve the model we must provide sufficient input data to constrain it. The data given in this example are typed into the input column of the variable sheet for *vol*, *thick*, *dens*, and *D*. The function key F9 will solve the model, and the results should appear in the output column beside the variables as seen in Table 2. Try it.

7 Comments may be typed into the comment column or not as you wish. They serve the useful purpose of documenting the model, which should always be done. The comments do not enter into the solution, however. The labels in the units column have special meanings when used in conjunction with a **units sheet**. We will introduce that powerful feature of *TK* in a later example. If you are typing in this model from scratch, then it is better to leave the units labels out for now.

8 The only caution that ALWAYS applies, whether using a units sheet or not, is that when first entering data to the variable sheet you must use only data that are **defined in a consistent unit system**. We have chosen *cgs* units in this example. Having made that choice we had to be consistent in using only data measured in that system,

Table 1 TKSolver Rule Sheet for Example 1 from File EX-01.tkw

```
; 1- calculate the volume of the cylinder                        (Eq. a)
        vol = basearea * H
; 2 - calculate the base area of the cylinder                    (Eq. b)
        basearea = PI() * D^2 / 4
; 3 - calculate the total surface area of the cylinder           (Eq. c)
        surface = basearea +  PI() * D * H
; 4 - calculate the weight of the cylinder                       (Eq. d)
        surface * thick * dens = weight
```

i.e., cm, g, and cc in this case. We will have a great deal of freedom to change the units of any variable after its first entry, and with a proper units sheet, *TK* will automatically convert them! But that's getting ahead of the story. (See Example 9 for a discussion of units conversion.)

Let's examine this model more closely. Note in Table 1 that equation (*a*) computes volume as a function of the base area. But the base area is not calculated until step 2. This arrangement of equations would not work in any conventional programming language or in most other equation solvers. They require any variable used on the right side of an equation to be defined numerically in advance of the execution of the equation in which it appears. All other programming languages and equation solvers are **procedural solvers**, meaning that they *proceed linearly through a set of instructions, from top to bottom, evaluating each expression as they go* (unless redirected by control statements, such as loops, if statements, etc.).

TKSolver operates differently. It is a **declarative solver**, which in simple terms means that it is *capable of sorting out the order in which a set of equations must be solved and reordering them* (in its "head") to solve them (if possible) no matter in what order you present them to it. We will see in a later example that it is possible to make *TK* behave in a procedural manner when we wish to, as when controlling a loop structure to manipulate an array, for example. Thus, *TK* is able to figure out that it needs to evaluate the second equation for *basearea* before it can solve the first equation for *vol*. It does this by making multiple passes through the rule sheet until it sorts out these hierarchies, and it will ultimately solve the model if enough data have been supplied to constrain it.

Equations (*b*) and (*c*) in Table 1 use a built-in function *PI( )* which returns the value of π to as many significant figures as the computer is capable of carrying (typically >15). The ( ) are required for proper syntax and, for most functions, the ( ) will contain one or more arguments to be passed to the function (e.g., *SIN(x)*); but *PI( )* does not need

Table 2 TKSolver Variable Sheet for Example 1 from File EX-01.tkw

St	Input	Variable	Output	Unit	Comments
	1 000	*vol*		cm^3	volume of container
	0.10	*thick*		cm	thickness of wall
	7.75	*dens*		g/cc	density of material
	12	*D*		cm	diameter of base of cylinder
		H	8.8	cm	height of side of cylinder
		basearea	113.1	cm^2	area of base
		surface	446.4	cm^2	total surface area
		weight	346	g	weight of empty cylinder

an argument, since it just returns a constant. Using this function is preferable to typing 3.1416 for π, as it gives more accuracy.

Look at equation (*d*) in Table 1. It is written in *implicit* form, in that the left side has more than one term. This is not allowed in any procedural solver or programming language. That simple equation could easily have been written with the single term *weight* on the left side, which would make it acceptable to a procedural solver. However, one often encounters complicated equations that either cannot be put into explicit form for a particular variable or require a great deal of algebraic manipulation in order to do so. In such instances this ability of *TK* to handle an implicit form is very useful. It was done in this example only to make this point.

Switching Variables from Input to Output and Vice Versa

We now redo the previous example with a different parameter specified as the input.

EXAMPLE 2

Simple Equation Solving Using TKSolver - Part B

Problem **Find the diameter and weight of an open-top, cylindrical container (a cooking pot) for a desired volume.**

Given **The volume = 1 liter, inside height = 10 cm, wall thickness = 0.1 cm.**

Assumptions **The material is stainless steel with a mass density of 7.75 g/cc.**

Solution **See Figure 1, Table 3, and the *TKSolver* file EX-02.tkw.**

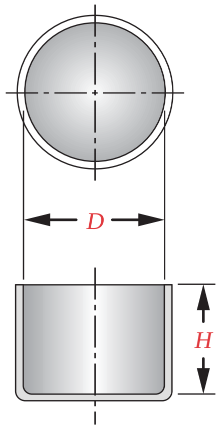

FIGURE 1

A Stainless Steel Cylindrical Container (Cooking Pot)

1 The only difference between this example and the previous one is that we have specified the pot's height and asked for the diameter needed to match a given volume. The rule sheet is thus the same as before. The model is unchanged, but we want to solve for a different variable.

2 Using *TKSolver*, this becomes a trivial issue. Table 3 shows the variable sheet for this example. Note that the value for *H* is now defined in the input column and the value of *D* has been computed in the output column. This switch was accomplished by simply typing the letter I (for input) in the leftmost column (labeled *St* for status) opposite the variable *H*. The I does not show, but it pulls the value into the input column. Then type a number 10 in the input column for *H*. Typing a letter O in the *St* column opposite *D* pushes its value to the output column. We have thus switched the status of these two variables. (Note that simply typing the number in the input column of *H* will change its status to input without requiring an *I* in the *St* column.)

3 Solving the model with function key F9 gives the results shown in Table 3.

Table 3 TKSolver Variable Sheet for Example 2 from File EX-02.tkw

St	Input	Variable	Output	Unit	Comments
	1 000	*vol*		cm^3	volume of container
	0.10	*thick*		cm	thickness of wall
	7.75	*dens*		g/cc	density of material
O*		*D*	11.3	cm	diameter of base of cylinder
I*	10	*H*		cm	height of side of cylinder
		basearea	100	cm^2	area of base
		surface	454.5	cm^2	total surface area
		weight	352.3	g	weight of empty cylinder

* These letters are not visible on the variable sheet, but change the status of the variable to input (I) or output (O).

Note the ease with which variables can be switched from input to output status or vice versa. This allows any variable to be solved for quickly and easily.

Using Iteration and Root Finding

We now complicate the problem by introducing a **height-to-diameter ratio** constraint (*H/D*) which will then require an iterative root-finding solution to a set of simultaneous equations.

EXAMPLE 3

Simultaneous Equation-Solving Using TKSolver

Problem **Find the diameter, height, and weight of an open-top, cylindrical container (a cooking pot) for a desired volume and height/diameter ratio (*H/D*).**

Given **The volume = 1 liter, *H/D* = 0.6, wall thickness = 0.1 cm.**

Assumptions **The material is stainless steel with a mass density of 7.75 g/cc.**

Solution **See Figure 1, Tables 4, 5, and 6, and the *TKSolver* file EX-03.tkw.**

1 This example introduces a new constraint to the problem posed in the previous two examples, but is otherwise the same. Instead of specifying either the diameter or height, we now specify the aspect ratio between those two parameters, and we want to find some combination of height and diameter that satisfies the two constraints of volume and aspect ratio. Define an additional rule that involves the aspect ratio.

$$ratio = \frac{H}{D} \tag{e}$$

2 Add this equation to the rule sheet as shown in Table 4 and push both *H* and *D* to the output column by typing an O in the status column as described in the previous example. Type the given value of 0.6 in the input column of the variable *ratio*.

3 Use Function key F9 to try solving the model now. You will find that it does not solve, because there are now two unknowns, *H* and *D*. An unsolved model is evidenced by the presence of asterisks to the left of the rules. When a rule is satisfied, the asterisk is erased.

4 We have two equations in these two unknowns, so they can be solved simultaneously. Equations (*a*) and (*b*) could be combined to give one expression for volume as *f*(*H*, *D*), and equation (*e*) expresses the ratio as *f*(*H*, *D*). Note that it is not necessary to rewrite the rule sheet to algebraically combine equations (*a*) and (*b*). The solver will "combine" them numerically.

5 To solve simultaneous, nonlinear equations it is necessary to provide guess values for one or more variables so that the Newton-Raphson root-finding algorithm* built into *TKSolver* can iterate to a solution. There are two ways to provide a guess value for any variable in *TK*. If you type a G in the *St* column next to the variable, it will use whatever number you type in the input column as an initial guess. If that guess is close enough to one of the roots of the equation system, it will converge to that root. Be aware, however, that nonlinear equations can have multiple roots, meaning that your solution can be different for different guess values.

6 The second (and preferred) way to provide a guess value for a variable is to type it on the variable's subsheet. To access the subsheet, place the cursor on the line containing the variable, pull down the WINDOW menu and select DISPLAY SUBSHEET. (This selection has the keyboard equivalent of *Alt* + *Enter.*) Once the variable's subsheet is open, any desired guess value can be typed on the line so labeled. The difference between these two methods of establishing a guess value is that a guess placed on the variable's subsheet will remain there through multiple solves of the

Table 4 TKSolver Rule Sheet for Example 3 from File EX-03.tkw

; 1- calculate the volume of the cylinder	(Eq. *a*)
*vol = basearea * H*	
; 2 - calculate the base area of the cylinder	(Eq. *b*)
*basearea = PI() * D^2 / 4*	
; 3 - calculate the total surface area of the cylinder	(Eq. *c*)
*surface = basearea + PI() * D * H*	
; 4 - calculate the weight of the cylinder	(Eq. *d*)
*surface * thick * dens = weight*	
; 5 - calculate the ratio of cylinder height to diameter	(Eq. *e*)
ratio = H / D	

* For information on root-finding algorithms, see *Numerical Recipes* by Press et al., Cambridge University Press.

Table 5 TKSolver Variable Sheet for Example 3 - Before Solving

St	Input	Variable	Output	Unit	Comments
	1 000	*vol*		cm^3	volume of container
	0.10	*thick*		cm	thickness of wall
	7.75	*dens*		g/cc	density of material
	0.6	*ratio*			height-to-diameter ratio
G	2	*D*		cm	diameter of base of cylinder
		H		cm	height of side of cylinder
		basearea		cm^2	area of base
		surface		cm^2	total surface area
		wt		g	weight of empty cylinder

model. But a guess value placed in the input column of the variable sheet along with a G in the *St* column will convert to an output value after solving and both the G and the guess value will have to be retyped for each subsequent solve.

7 Use either of these methods to define a guess value (the supplied example file has a guess value of 2 on *D*'s subsheet) and solve the model with F9. It will automatically iterate to the solution. Table 5 shows the variable sheet for this example with the G shown in the *St* column and the guess value of 2 in the input column *before solution*. Table 6 shows the same variable sheet after solution. The solver has iterated to the simultaneous solution that satisfies the constraints on both volume and ratio to give the values of *H* and *D* shown in the output column.

Table 6 TKSolver Variable Sheet for Example 3 - After Solving

St	Input	Variable	Output	Unit	Comments
	1 000	*vol*		cm^3	volume of container
	0.10	*thick*		cm	thickness of wall
	7.75	*dens*		g/cc	density of material
	0.6	*ratio*			height-to-diameter ratio
		D	12.9	cm	diameter of base of cylinder
		H	7.7	cm	height of side of cylinder
		basearea	129.7	cm^2	area of base
		surface	441	cm^2	total surface area
		wt	341.8	g	weight of empty cylinder

The previous three examples show how *TKSolver* can quickly solve individual or simultaneous equations. Variables can be switched from input to output status allowing a model to be solved for any parameter present in its rules without requiring any rewriting of those rules. Iterative root finding is automatically invoked if the model cannot be directly solved and if a sufficient number of guesses for the unknown variable values have been provided, either on the variable sheet or on the variables' subsheets.

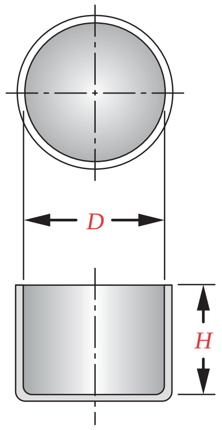

FIGURE 1

A Stainless Steel Cylindrical Container (Cooking Pot)

Lists, Tables, Plots, Optimization, and Built-in Functions

The next two examples introduce the use of lists (arrays) in *TKSolver* and show how an optimum solution to a problem can be quickly found. Once lists of variables are created, plots and tables of the model's parameters can also be quickly generated.

EXAMPLE 4

List-Solving and Plotting Using TKSolver

Problem **Find the height/diameter ratio and dimensions of an open-top, cylindrical container that will minimize its weight for a given volume. Plot the variation of weight with the height/diameter ratio.**

Given **Volume = 1 liter, wall thickness = 0.1 cm.**

Assumptions **The material is stainless steel with a mass density of 7.75 g/cc. The ratio *H/D* will be varied from 0.1 to 2.0 in increments of 0.1.**

Solution **See Figures 1 and 2, Tables 7, 8, and 9, and the *TKSolver* file EX-04.tkw.**

1 There should be an optimum *H/D* ratio for this container that will minimize its surface area and weight. Since the material is sold by weight, its cost will then be minimized as well. This simple example could be optimized by writing an expression for the weight as a function of *H/D* ratio, differentiating it with respect to that ratio, setting the differential equal to zero, and solving for the ratio. However, we can also obtain a numerical approximation to that optimum ratio from the existing *TKSolver* model by creating lists of variables and *list solving* the model for all values in the input list.

2 A variable can be made into a list simply by typing a letter L in the status column of that variable as shown in Table 7. The variables ratio, *D*, *H*, and weight have been so designated.

3 At least one list must be declared as an input list and a set of input values provided to it. The variable *ratio* is made to be the input list simply by the fact that it has a value in its input column on the variable sheet. Note that all the other list variables' values are in the output column.

Table 7 TKSolver Variable Sheet for Example 4 from File EX-04.tkw

St	Input	Variable	Output	Unit	Comments
	1 000	*vol*		cm^3	volume of container
	0.10	*thick*		cm	thickness of wall
	7.75	*dens*		g/cc	density of material
L	0.6	*ratio*			height-to-diameter ratio
L		*D*	12.9	cm	diameter of base of cylinder
L		*H*	7.7	cm	height of side of cylinder
		basearea	129.7	cm^2	area of base
		surface	441	cm^2	total surface area
L		*weight*	341.8	g	weight of empty cylinder

4 To put values in the input list, pull down the COMMANDS menu and select LIST FILL. The resulting dialog box allows the desired first, last, and step values for the list to be specified, and fills the list. To view its contents, place the cursor on the variable-sheet line containing that variable and use *Alt* + *Enter* twice in succession to "dive" down two levels into the sub-subsheet for that variable. This will expose the list in its own window. The list for the variable *ratio* was filled by this technique with 20 values from 0.1 to 2.0 in steps of 0.1 and is shown in Table 8.

5 Use the function key F10 to do a **list solve**. This performs a separate solution of the rules for each value in the input list. In this example it solved the rules 20 times. The output lists *D*, *H*, and *weight* are now filled with 20 values apiece, each one corresponding to a solution for one of the values in the input list *ratio*. In terms of a conventional programming language such as BASIC, Fortran, Pascal or C, the list-solve operation has, in effect, executed a loop which repeated the rule-sheet calculations while incrementing through all the values in the input list.

6 To see the results in convenient form, we can create an **interactive table**. Open the **table sheet** from the WINDOW pull-down menu. Type any name you wish in the column labeled "name." A title for the table also can be typed in the title column if desired, or it can be left blank. With the cursor sitting on the line in the table sheet containing the name you just typed, "dive" (*Alt* + *Enter*) to go to the subsheet for this table. In the subsheet column headed "List," type the name of any list variable that you want to put in the table. In this example, we typed *ratio*, *D*, *H*, and *weight*, each on a new line in the List column. To display the table "dive" again using *Alt* + *Enter.* The **interactive table** will now be visible in its own window and should look like Table 9. It can be scrolled through on-screen and can be printed. Since it is an "interactive" table, any changes typed into it will automatically change the parent lists. For example, if you wanted to change the last value of the input list *ratio* from 2.0 to 3.0, typing the new value in the interactive table will have the same effect as if you typed it directly into the list. They are "hot-linked."

Table 8
TKSolver List for the Variable *ratio*

Element	Value
1	0.1
2	0.2
3	0.3
4	0.4
5	0.5
6	0.6
7	0.7
8	0.8
9	0.9
10	1.0
11	1.1
12	1.2
13	1.3
14	1.4
15	1.5
16	1.6
17	1.7
18	1.8
19	1.9
20	2.0

7 A plot of any list versus any other list (or lists) can also be quickly created. Open the plot sheet from the WINDOWS menu. Type any name you wish in the column labeled "name." A title for the plot also can be typed in the title column if desired, or it can be left blank. With the cursor sitting on the line in the plot sheet containing the name you just typed, "dive" (*Alt + Enter*) to go to the subsheet for this plot. The cursor will be on a field labeled "X-Axis List." Type the name of the list that you want plotted on the *x* axis as the independent variable. For our example this is the list *ratio*. Then place the cursor on one of the fields in the column under the label "Y-Axis" and type the name of a list that you want plotted as a dependent variable on the *y* axis. For our example this could be the list *weight*. Use the function key F7 to display the plot. It should look like Figure 2.

8 The optimum solution is obvious from Figure 2. The weight of the container is a minimum at an *H/D* ratio of about 0.5. Table 9 shows this weight to be 340.5 g.

The preceding example shows how easy it is to obtain plots and tables of solution data and from them determine an approximate optimum design. To do this without a tool such as an equation solver would be tedious indeed. Even to write a custom computer program in Pascal, BASIC, or C would be more time consuming than this approach. An equation solver eliminates the "programming overhead" associated with input and output of data, table and plot generation, etc. Solutions to even simple problems like this example can be obtained in less time than with most other methods. When the problem is more complex, an equation solver becomes an indispensable tool for the engineer.

Using Built-in Functions

We now modify the *TKSolver* file from the preceding example to add the ability to extract the optimum solution values from the lists without the need for a table or plot (though both of those displays will still be of value in visualizing the solution). In the process, we also increase the size of the lists from their present 20 to 80 elements in order to get a closer approximation to the optimum solution.

EXAMPLE 5

List Solving and Built-in Functions in TKSolver

Problem **Find the height/diameter ratio and dimensions of an open-top, cylindrical container that will minimize its weight for a given volume. Plot the variation of weight with the height/diameter ratio. Extract the optimum values from the lists of solution variables.**

Given **Volume = 1 liter, wall thickness = 0.1 cm.**

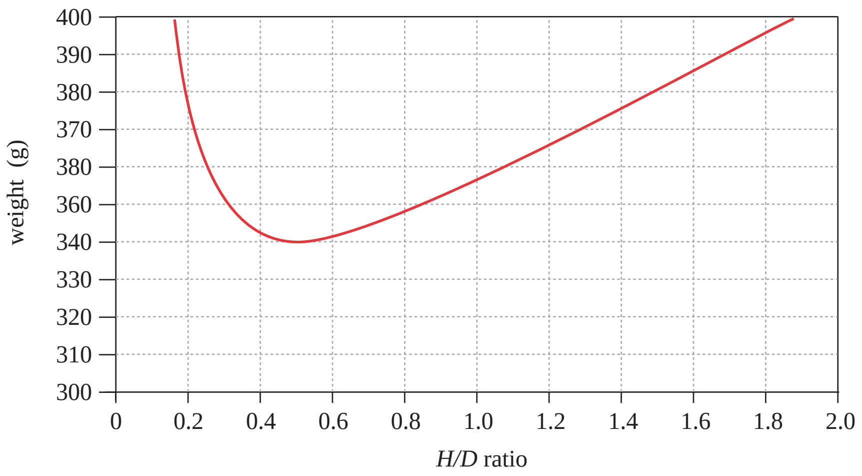

FIGURE 2

Variation of Weight with Height-to-Diameter Ratio in Example F-4

Table 9 TKSolver Interactive Table for Example 4

Element	ratio	D	H	weight
1	0.1	23.4	2.3	464.7
2	0.2	18.5	3.7	376.4
3	0.3	16.2	4.9	351.1
4	0.4	14.7	5.9	342.5
5	0.5	13.7	6.8	340.5
6	0.6	12.9	7.7	341.8
7	0.7	12.2	8.5	344.7
8	0.8	11.7	9.3	348.5
9	0.9	11.2	10.1	352.9
10	1.0	10.8	10.8	357.5
11	1.1	10.5	11.5	362.4
12	1.2	10.2	12.2	367.3
13	1.3	9.9	12.9	372.2
14	1.4	9.7	13.6	377.1
15	1.5	9.5	14.2	382.0
16	1.6	9.3	14.8	386.8
17	1.7	9.1	15.4	391.6
18	1.8	8.9	16.0	396.3
19	1.9	8.8	16.6	400.9
20	2.0	8.6	17.2	405.4

Assumptions **The material is stainless steel with a mass density of 7.75 g/cc. The ratio *H/D* will be varied from 0.1 to 2.0 in increments of 0.025.**

Solution **See Tables 10 and 11 and the *TKSolver* file EX-05.tkw.**

1 Fill the input list *ratio* with 77 elements with values from 0.1 to 2.0 in increments of 0.025. See step 4 in Example 4 for the procedure.

2 *TKSolver* has about 100 built-in functions to perform common mathematical operations such as square roots, logarithms, trigonometric operations, etc. Their syntax can be viewed with the on-line help in *TKSolver*, either from the HELP pull-down menu, or by using the function key F1. We will use three of these functions in this example to demonstrate function use in general and also to show how to extract information from lists.

3 Table 10 shows the rule sheet for this example. Three rules have been added to those of Example 4. At line 6, the *MIN*(*x*) function is used to return the smallest value in the list '*weight*. The single quote in front of *weight* indicates that it is the name of a list rather than a variable for which a single value exists on the variable sheet. Equation (*f*) in Table 10 places the minimum value returned from this function in the variable *minwt*. The variable sheet in Table 11 shows the value it returned to *minwt* after solving. Note a subtlety here. A list solve (F10) must first be done in order to fill the list '*weight* with its calculated values. Then a direct solve (F9) also must be done to cause equation (*f*) to extract the minimum value from that list.

4 We now have found an approximation of the lowest weight for our design, but we really need to know the value of the *H/D* ratio that is responsible for that minimum

Table 10 **TKSolver Rule Sheet for Example F-5 from File EX-05.tkw**

; 1- calculate the volume of the cylinder	(Eq. *a*)
*vol = basearea * H*	
; 2 - calculate the base area of the cylinder	(Eq. *b*)
*basearea = PI() * D^2 / 4*	
; 3 - calculate the total surface area of the cylinder	(Eq. *c*)
*surface = basearea + PI() * D * H*	
; 4 - calculate the weight of the cylinder	(Eq. *d*)
*surface * thick * dens = weight*	
; 5 - calculate the ratio of cylinder height to diameter	(Eq. *e*)
ratio = H / D	
; 6 - find the minimum value in the list 'weight	(Eq. *f*)
minwt = MIN('weight)	
; 7 - find the location (subscript) of the min value in the list 'weight	(Eq. *g*)
locmin = MEMBER(minwt, 'weight)	
; 8 - find the value in the list 'ratio with the subscript locmin	(Eq. *h*)
minratio = ELEMENT('ratio, locmin)	

weight. The function *MEMBER*(*minwt*, ‘*weight*) at line 7 returns the location (or subscript) in the array ‘*weight* of the element that has the value *minwt*. This value is shown as *locmin* in Table 11.

5 We can now use the function *ELEMENT*(‘*ratio*, *locmin*) at line 8 to return the value of ‘*ratio*[*locmin*]. This is the desired solution, i.e., the value of the *H/D* ratio that will give the minimum-weight container.*

6 Compare these results with Table 9 and the plot in Figure 2.

It is worth the effort to spend a little time looking at the collection of built-in functions provided with the software. Their use can save programming time. See the *TKSolver* manual and its on-line help for more information.

User-Defined Functions

One of the most powerful features of *TKSolver* is its ability to accommodate user-defined functions. Three types of user functions are available: **rule functions**, **procedure functions,** and **list functions**. Each has a different purpose and set of applications. We present an example of each in the order listed above.

RULE FUNCTIONS A rule function performs like the rule sheet but allows a set of rules to be isolated within a callable function that has its own set of local variables. Its local variables are isolated from those on the variable sheet. Values are passed into the rule function as **argument variables** when it is called either from the rule sheet or from

Table 11 TKSolver Variable Sheet for Example 5 from File EX-05.tkw

St	Input	Variable	Output	Unit	Comments
	1 000	*vol*		cm^3	volume of container
	0.10	*thick*		cm	thickness of wall
	7.75	*dens*		g/cc	density of material
L	0.6	*ratio*			height to diameter ratio
L		*D*	12.9	cm	diameter of base of cylinder
L		*H*	7.7	cm	height of side of cylinder
		basearea	129.7	cm^2	area of base
		surface	441	cm^2	total surface area
L		*weight*	341.8	g	weight of empty cylinder
		minwt	340.5	g	minimum weight for ratio
		locmin	17		array location of *minwt*
		minratio	0.5		optimum *H/D* ratio for *minwt*

* Note that this problem could have been solved quickly and easily for the exact minimum weight using the calculus. We use a less-exact numerical method here to show the process. A numerical method is often the only possible solution to more complicated problems.

another function. Its computed values are passed back to the calling program as result variables. Within a rule function, the declarative solver is in operation, so the order of equations is not critical and implicit rules are acceptable. Anything that can be done on a **rule sheet** can be done in a **rule function**.

PROCEDURE FUNCTIONS A procedure function is very similar to a procedure in PASCAL or a subroutine in FORTRAN. It has local variables, input and output variables, and is activated with a CALL statement. Its execution is procedural, meaning its statements are executed in linear fashion, top to bottom, unless redirected with control statements. It behaves like any procedural programming language. All its statements must be explicit with only one variable on the left side. Its purpose is to allow looping through lists and/or execution of control structures that are not possible within the declarative solver environment.

LIST FUNCTIONS are table-look-up functions that allow several types of mapping between their input and output lists, including two forms of interpolation. List functions prove to be extremely useful in design problems where much information needed for the solution is discrete in nature, such as material strengths for various alloys, or stock sizes available for fasteners, wire, or I-beams, etc. Empirical data generated from tests can be encoded as list functions and looked up "on-the-fly" while a model's rules are iterating to a solution using guess values for other continuous variables.

Rule Functions

We now modify the previous example to introduce the use of a **rule function**. Subsequent examples will use procedure and list functions as well.

EXAMPLE 6

Using Rule Functions in TKSolver

Problem **Use a rule function to find the height/diameter ratio and dimensions of an open-top, cylindrical container that will minimize its weight for a given volume. Plot the variation of weight with height/diameter ratio.**

Given **Volume = 1 liter, wall thickness = 0.1 cm.**

Assumptions **The material is stainless steel with a mass density of 7.75 g/cc. The ratio *H/D* will be varied from 0.1 to 2.0.**

Solution **See Tables 12, 13, and 14 and the *TKSolver* file EX-06.tkw.**

1 Table 12 shows the rule function *cyl_rule** which was created to solve the first four equations (*a* - *d*) in the rule sheet for Example 4. To do this, open the **function sheet** from the WINDOW menu and type the desired function name (*cyl_rule*) in the column

Table 12 TKSolver Rule Function for Example 6 from File EX-06.tkw

Rule Function cyl_rule

Comment:	Calculates volume and area of cylinder
Parameter variables:	
Argument variables:	V, ratio
Result variables:	dia, height, A

S **Rule**

```
; Rule Function to calculate surface area and volume of a cylindrical container
; to use: CALL cyl_rule ( V, ratio ; dia, height , A )
        ratio = height / dia
        basearea = PI() * dia^2 / 4
        A = basearea +  PI() * dia * height
        V = basearea * height
```

labeled "Name." In the column labeled "Type," put the letter *R* to identify it as a **rule function**. A comment is optional. Dive on this line to open the actual rule function subsheet shown in Table 12.

2 Type the desired equations into the rule function as they would appear on a rule sheet. In fact, the equations from an existing rule sheet such as that of Example 4 can be copied to the clipboard and pasted into the rule function to save typing time. Note in Table 12 that different variable names have been used for some of the parameters. This is arbitrary and was done to make the point that these are local variables, different from those on the rule sheet.

3 At the top of the rule function are three lines labeled **parameter variables**, **argument variables**, and **result variables**. **Parameter variables** have common values among the variable sheet and all functions in which they appear. Anything in this parameter list becomes a **global variable** and loses its local character. We do not need any of these in this function, so the line is left blank.

4 **Argument variables** are the input variables to the rule function. Type all of the variable names needed as input to solve the rules in the function on this line, separated by commas. In this example, only *V* and *ratio* are required as input. (Note that these are the only input values on the variable sheet of Table 7 that are used in these rules.)

* Note that the use of the underscore (_) in the name of the function is NOT required. Any word can be used for a function name as long as it does not contain certain characters that have mathematical meaning such as +, -, etc. The underscore is used here simply to increase readability of the function name. Variable and function names in *TKSolver* are "case sensitive," meaning that capitalization of any letter(s) changes the name, i.e., Cyl_rule is a different name than cyl_rule. This case sensitivity can cause some consternation when writing and debugging models, as one's eye may not notice that the "same" variable name has been inadvertently typed in two places with different case. *TKSolver* will consider them different variables and the model will probably not solve, because only one of them has a value associated with it on the variable sheet. This is one of the most common errors to check for when your model refuses to solve. See the *TKSolver* manual for complete information on naming functions and variables.

Table 13 TKSolver Rule Sheet for Example 6 from File EX-06.tkw

```
; 1 - calculate the dimensions and area of the cylinder using the rule function "cyl_rule"
        CALL cyl_rule ( vol, ratio ; D, H , surface )
; 2 - calculate the weight of the cylinder
        weight = surface * thick * dens
```

5 **Result variables** are the output from the function. Type any of the variables from the equations in the function for which you wish to return values to the calling program. Note that it is not necessary to put all of the equations' variables in these lists. Any variables not listed in one of these three lines (such as *basearea* in this example) will remain as local variables known only to this rule function.

6 The comments within the rule function preceded by a semicolon* are optional and serve to document the function. It is a good idea to document the required CALL statement that will be used in any rule sheet or other function to activate this function. The format of the CALL statement is as shown in Tables 12 and 13. Note that the argument variables (inputs), separated by commas, are listed first and are separated from the list of comma-delimited result variables with a semicolon.

7 The rule statement is now reduced to two statements as shown in Table 13. The first is the CALL statement (without the preceding ; or " so it will execute). Note that the names of the variables passed to the rule function do not have to be the same as the names used in the rule function. The variable names in the rule function are "dummies," each of which takes on the label of whatever is passed to it in the CALL statement. The mapping between the actual and dummy variables is by their **order** in the argument list, first to first, second to second, etc. In this case *vol* in the rule sheet (Table 13) becomes *V* in the rule function (Table 12), *ratio* is *ratio*, *dia* becomes *D*, etc. The second rule-sheet equation calculates the weight of the cylinder. This equation could have been placed inside the rule function if desired, which would then require *thick* and *dens* to be added to its argument list.

8 When this model is solved, it gives the same solution as in Example 4. The variable sheet is shown in Table 14. Note that the dummy variables *V*, *dia*, *height*, *A*, and *basearea* do not appear on the variable sheet. Only the global variables created in the rule sheet appear there.

9 The principal advantage of using rule (or other) functions is to encapsulate rule sets that may be useful in more than one program into a form that is easily transferred from one model to another. The localization of variables inside the function means that you can call the variables by different names in different models and still use the same rule function. Another advantage is that of modularizing the model, which means breaking it down into more tractable pieces that can be debugged and proven independently of the rest of the model. The value of this approach increases as your models get more complicated.

* The semicolon is used to denote a comment line in the Windows and DOS versions of *TKSolver*. The Macintosh version uses a quote (") instead of a semicolon for the same purpose.

Procedure Functions

We will now introduce the use of procedure functions to control access to lists in the following example.

Table 14 TKSolver Variable Sheet for Example 6 from File EX-06.tkw

St	Input	Variable	Output	Unit	Comments
	1 000	*vol*		cm^3	volume of container
	0.10	*thick*		cm	thickness of wall
	7.75	*dens*		g/cc	density of material
L	0.6	*ratio*			height-to-diameter ratio
L		*D*	12.9	cm	diameter of base of cylinder
L		*H*	7.7	cm	height of side of cylinder
		surface	441	cm^2	total surface area
L		*weight*	341.8	g	weight of empty cylinder

EXAMPLE 7

Using Procedure Functions in TKSolver

Problem **Create a procedure function to load the input list *ratio* in Example 6 so that the range of values for calculation can be defined by the user at run time. Use the rule function *cyl_rule* to find the height/diameter ratio and dimensions of an open-top, cylindrical container that will minimize its weight for a given volume.**

Given **Volume = 1 liter, wall thickness = 0.1 cm.**

Assumptions **The material is stainless steel with a mass density of 7.75 g/cc. The ratio *H/D* will be varied from 0.1 to 2.0 using 40 data points.**

Solution **See Tables 15, 16, and 17 and the *TKSolver* file EX_07.tkw.**

1 The task here is to automatically fill a list with *N* values that start at a user-defined minimum value (*min*) and end at a user-defined maximum value (*max*). The value of *N* will be user specified as well. We cannot do this on the rule sheet or in a rule function because the declarative solver will control the order in which the statements are executed. We need to control a loop structure and cause the desired numbers to be placed in the list one by one in order. This requires the sequential solving of a procedure function.

2 To create a procedure function, open the **function sheet** from the WINDOW menu and type the desired function name (here *filalist*) in the column labeled “Name.” In the column labeled “Type,” put the letter *P* to identify it as a procedure function. A comment is optional. Dive on this line to open the procedure-function subsheet.

3 There are two built-in functions that will help with this task. The BLANK (*listname*) function blanks the list whose ‘*name* is provided in the variable *listname*. The

Table 15 TKSolver Procedure Function for Example 7 from File EX-07.tkw

Procedure filalist

Comment:	Fills a list with range of values
Parameter Variables:	
Input Variables:	N,min,max,name
Output Variables:	

```
S       Statement
          ; N is the number of data in the list
          ; min is first value
          ; max is last value
          ; name is a string variable containing the name of the list
          ; to use:  CALL filalist ( N, min, max, 'listname )
          ; first blank the list
               CALL BLANK(name)
               T = min
               delta = (max - min) / (N - 1)
               FOR I = 1 to N
                    PLACE ( name, I ) = T
                    T = T + delta
               NEXT  I
```

PLACE (*listname*, *I*) = *x* function places the value of *x* in the Ith element of the list whose ‘*name* is provided on the variable sheet in the variable *listname*.

4 The finished procedure function is shown in Table 15. The input variables are listed on the line with that label. Their meanings are defined in the comment lines. The heart of this procedure is the FOR-NEXT loop which runs its index *I* from 1 to *N* and uses *I* as an argument in the built-in function PLACE(*name*,*I*) = *T*. This statement inserts the value of *T* into the *I*th element of the list “name.” The value of *T* is initialized to the value *min* and increased on each pass through the loop by *delta*.

Table 16 TKSolver Rule Sheet for Example 7 from File EX-07.tkw

```
; 1 - fill the input list called 'ratio with desired values
      CALL filalist ( N, min, max, 'ratio )
; 2 - calculate the area and volume of the cylinder using the rule function ;cyl_rule;
      CALL cyl_rule ( vol, ratio ; D, H , surface )
; 3 - calculate the weight of the cylinder
      weight = surface * thick * dens
```

Table 17 TKSolver Variable Sheet for Example 7 from File EX-07.tkw

St	Input	Variable	Output	Unit	Comments
	40	*N*			number of data points desired
	0.1	*min*			minimum value for input list
	2	*max*			maximum value for input list
L	0.5	*ratio*			height-to-diameter ratio
	1 000	*vol*		cm^3	volume of container
	0.1	*thick*		cm	thickness of wall
	7.75	*dens*		g/cc	density of material
L		*D*	13.7	cm	diameter of base of cylinder
L		*H*	6.8	cm	height of side of cylinder
		surface	439.4	cm^2	total surface area
L		*weight*	340.5	g	weight of empty cylinder

Delta is calculated from the values of *min*, *max*, and *N*. Note the similarity of this procedure to a BASIC computer program; the loop structure is identical.

5 The rule sheet that calls the procedure is shown in Table 16. The CALL to *filalist* passes the list name ‘ratio for loading with *N* values between *min* and *max*. This will have the same effect as manually loading the list with the LISTFILL menu command, but it is now automated. Open the list sheet to see that the list ‘ratio now has *N* values from *min* to *max* in it after solving the model. The rest of the rule sheet is identical to that of Example 6.

6 The variable sheet for this model is shown in Table 17. The values of *N*, *min*, and *max* are the only additions to the model of Example 6 shown in Table 14.

This is a somewhat trivial example of the use of a procedure function[*] but it does illustrate its ability to manipulate lists (arrays) in conventional programming fashion. Applications of procedure functions can be found in the examples and case studies presented in this book.

List Functions

We now add to our example a list function to select the density of the material based on the input of an alphameric code to the variable sheet. Again, this is a somewhat trivial application of the power of list functions, but be assured that these functions have great value in automatically selecting tabular data while calculating a model. List functions are used extensively in this book’s examples and case studies.

[*] It is also a crude implementation in that the CALL to the procedure *filalist* will be repeated (unnecessarily) each time the list solver performs an execution of the rule sheet. The only penalty in this case is a waste of execution time. The process could be speeded up by using an IF statement to limit the execution of *filalist* to the first pass through the rule sheet only. This was not done in this example in order to adhere to the principle of KISS (Keep It Simple, Stupid!).

Table 18 TKSolver List Function for Example 8 from File EX-08.tkw

List Function: *get_dens*

Comment:	Returns the weight density of a material
Domain List:	material
Mapping:	Table
Range List:	density

Element	Domain	Range
1	'alum	2.76805
2	'steel	7.75054
3	'copper	8.58096

EXAMPLE 8

Using List Functions in TKSolver

Problem **Create a list function to return the density of a selected material. Use that density value in combination with the rule function *cyl_rule* to find the height/diameter ratio and dimensions of an open-top, cylindrical container that will minimize its weight for a given volume.**

Given **Volume = 1 liter, wall thickness = 0.1 cm.**

Assumptions **The material is aluminum with a mass density of 2.77 g/cc. The height-to-diameter ratio *H/D* will be varied from 0.1 to 2.0.**

Solution **See Tables 18, 19, and 20 and the *TKSolver* file EX-08.tkw.**

1 A list function relates the contents of two lists. The input list is called the **domain** of the function and the output list is called its **range**. The input value is passed to the list function as an argument and it returns the corresponding output value.

Table 19 TKSolver Rule Sheet for Example 8 from File EX-08.tkw

```
; 1 - fill the input list called 'ratio with desired values
        CALL filalist ( N, min, max, 'ratio )
; 2 - retrieve the density of the chosen material from the list function “get_rule"
        dens = get_dens ( material )
; 3 - calculate the dimensions and area of the cylinder using the rule function "cyl_rule"
        CALL cyl_rule ( vol, ratio ; D, H , surface )
; 4 - calculate the weight of the cylinder
        weight = surface * thick * dens
```

2 To create a list function, open the **function sheet** from the WINDOW menu and type the desired function name (here *get_dens*) in the column labeled "Name." In the column labeled "Type," put the letter *L* to identify it as a list function. A comment is optional. Dive on this line to open the list-function subsheet.

3 Table 18 shows this list function. When making your own, start by typing, on the line beside the label "Domain List," the name of the list that you want to use as the input. This list name can be one that already exists on the list sheet or can be a new list, still to be filled. We called the domain list *material* for this example.

4 For this example we want **Table** mapping, which creates a one-for-one correspondence between input and output variables with no interpolation. Other possible mappings are **step**, **linear interpolation,** and **cubic interpolation**, See the *TKSolver* on-line help for further information on these.

5 Type the name of the desired output list on the line labeled "Range List." Here we used the list name *density*.

6 If these lists have already been defined on the list sheet and filled with data, they will immediately appear in the columns headed Domain and Range. If they have not yet been filled, you can fill them from this sheet. Simply type the appropriate words or numbers on each element's line. In this simple example we want some convenient labels for the materials in the domain list (*matl*) and the corresponding densities of those materials in the range list (*density*). To create your list function, type the data as shown in Table 18, using the single quote on the names to designate them as alphanumeric data.

7 The rule sheet is shown in Table 19. At line 2, the list function is called by using it in the assignment statement:

$$dens = get\_dens(material) \qquad (f)$$

8 Note on the variable sheet in Table 20 that the variable *material* is set to the value 'alum, which is one of the labels in the list *matl*. When the rule (*f*) is evaluated, it passes the value 'alum to the list function *get_dens*, which scans its domain list for that value. If it finds an 'alum in its domain, it returns the corresponding value from the range list (2.76805) and places it in the variable *dens* on the variable sheet. Note that neither of the list variables *matl* or *density* that belong to the list function appears on the variable sheet.

9 The solution then proceeds as before, using the returned value of *dens* which is now an output variable.

Units and Formatting

TKSolver allows automatic unit conversions within any model that possesses a **units sheet**. The units sheet must be created by the user. A fairly extensive one is supplied on disk with this text as the file UNITMAST. A small portion of this unit sheet is shown as Table 21. This file can be merged into any *TK* model by using the MERGE command on the FILE pull-down menu. Once merged into the model, typing any of this sheet's

Table 20 TKSolver Variable Sheet for Example 8 from File EX-08.tkw

St	Input	Variable	Output	Unit	Comments
	40	*N*			number of data points desired
	0.1	*min*			minimum value for input list
	2	*max*			maximum value for input list
L	0.5	*ratio*			height-to-diameter ratio
	1 000.	*vol*		cm^3	volume of container
	0.1	*thick*		cm	thickness of wall
	'alum	*material*			one of 'alum, 'steel, or 'copper
		dens	2.77	g/cc	density of material
L		*D*	13.7	cm	diameter of base of cylinder
L		*H*	6.8	cm	height of side of cylinder
		surface	439.4	cm^2	total surface area
L		*weight*	121.6	g	weight of empty cylinder

unit abbreviations in the units column of the variable sheet will invoke conversion of that variable's units, provided that a conversion factor exists on the units sheet. Formatting of variables is also possible by using the **format sheet**. The use of both units and format sheets will be demonstrated in an example.

Table 21 Part of the TKSolver Units Sheet from the File UNITMAST

From	To	Multiply By	Add Offset	Comment
lb	N	4.448 2		force
N	dyne	100 000		force
m	cm	100		length
in	cm	2.54		length
in^2	mm^2	645.162 6		area
m^2	mm^2	1 000 000		area
cm^2	mm^2	100		area
m^3	mm^3	1E+09		volume
cm^3	mm^3	1 000		volume
cm^3	cc			volume
g/cc	g/mm^3	0.001		density
kg/m^3	g/mm^3	0.000 001		density
lb/in^3	g/cc	27.680 5		density
lb/ft^3	lb/in^3	1 728		density
C	F	1.8	32	temperature

EXAMPLE 9

Using Unit Sheets and Format Sheets in TKSolver

Problem **Convert the *cgs* units used in the previous example to the *SI* system.**

Solution **See Tables 21, 22, 23, and 24 and the *TKSolver* file EX-09.tkw.**

1 The previous examples were all calculated in the *cgs* units system. The *SI* system is generally preferred for metric units. With the units sheet shown in Table 21 present in the *TK* file, the conversions to *SI* units are accomplished simply by typing the desired unit symbols from Table 21 into the units column of any variable to be converted. The result is shown in Table 22. If a unit symbol that does not exist on the units sheet is typed, it will not convert the variable's value but will instead put a question mark in front of its value to indicate that the conversion is not possible with this unit sheet. If more than one symbol is desired for a given variable, such as cc and cm^3 (or cm3) for cubic centimeters, this can be accomplished by providing a unity conversion factor between the equivalent symbols as was done in Table 21.

2 If any variable's subsheet is opened (by diving on its line in the variable sheet), it will look like that shown in Table 23 for the variable *D*. Detailed information about the variable is recorded here. In this case, the status line indicates *D* is a list variable. It has a guess value of 0.02 assigned to it, which will be used whenever the **direct solver** cannot resolve the rules and must invoke the **iterative solver**. Putting a guess value here eliminates the need to repeatedly put a G in the status column on the variable sheet when solving. Since this is an output variable at present, the input-value line is blank and its calculated value appears on the output-value line.

3 Note the two lines labeled **Display Unit** and **Calculation Unit**. The first of these defines the units currently displayed on the variable sheet and the second defines the

Table 22 TKSolver Variable Sheet for Example 9 from File EX-09.tkw

St	Input	Variable	Output	Unit	Comments
	40	*N*			number of data points desired
	0.1	*min*			minimum value for input list 'ratio
	2	*max*			maximum value for input list 'ratio
	0.001	*vol*		m^3	volume of container
	0.001	*thick*		m	thickness of wall
	alum	*material*			one of 'alum, 'steel, or 'copper
		dens	2 768.	kg/m^3	density of material
L		*D*	0.137	m	diameter of base of cylinder
L		*H*	0.068	m	height of side of cylinder
L	0.5	*ratio*			height-to-diameter ratio
		surface	0.044	m^2	total surface area
L		*weight*	0.122	kg	weight of empty cylinder

Table 23 TKSolver Variable Subsheet for Example 9 from File EX-09.tkw

Variable: D	
Status:	L
First Guess:	0.02
Associated List:	D
Input Value:	
Output Value:	0.137
Numeric Format:	d3
Display Unit:	m
Calculation Unit:	cm
Comment:	Diameter of base of cylinder

unit in which all calculations of this variable are done. **It is critical that all variables in the model have calculation units consistent with one of the standard units systems**. In this case, the calculation unit is cm, because that was the unit label first typed into the variable sheet in Example 1-2. **Whatever unit is first typed on the variable sheet becomes the calculation unit and will remain so unless changed on each variable's subsheet**. **Any subsequent changes to the unit label on the variable sheet change only the display unit**. This is why it is so important

Table 24 TKSolver Format Sheet for Example 9 from File EX_09.tkw

Format: d3	
Comment:	3 decimal places
Numeric Notation:	Decimal
Significant Digits:	18
Decimal Places:	3
Padding:	Zero
Decimal Point Symbol:	.
Digit Grouping Symbol:	,
Zero Representation:	0.000
+/– Notation:	– Only
Prefix:	
Suffix:	
Justification:	Right
Left Margin Width:	0
Right Margin Width:	0

to be consistent with one units system when first applying unit labels to a variable sheet. Failure to do so may lead to erroneous numerical results. You can have any mix of display units on the variable sheet and this will not affect the correctness of the results as long as the calculation units for all variables are in a consistent units system.

4 The **Numeric Format** line allows specification of a format for this variable provided that the format code (*d3*) is defined on the format sheet. Table 24 shows the format subsheet for this *d3* specification. The user can make up any set of format codes and define their attributes on this sheet. The *TK* file FORMATS supplied on disk has a format sheet that defines useful decimal and exponential formats. This file can be merged into any *TK* model. A default global format for the entire variable sheet can also be defined from the SETTINGS menu found under the FORMAT pull-down menu. The file STUDENT contains both the units sheet from UNITMAST and the format sheet from FORMATS.

EXERCISES

1 Write a *TKSolver* program to calculate the cross-sectional properties for all the shapes shown in Appendix A of reference 1.

2 Convert the program in Problem 1-7 to a set of rule functions, one for each shape in Appendix A of reference 1. Each function should return the area and second moments of area for one section shape.

3 Write a *TKSolver* program to calculate the mass properties for all the solids shown in Appendix B of reference 1.

4 Convert the program in Problem 1-9 to a set of rule functions, one for each solid in Appendix B of reference 1. Each function should return the volume and second moments of mass for one section shape.

REFERENCES

1 Norton, R. L., *Machine Design: An Integrated Approach* 3ed. Prentice-Hall, 2005