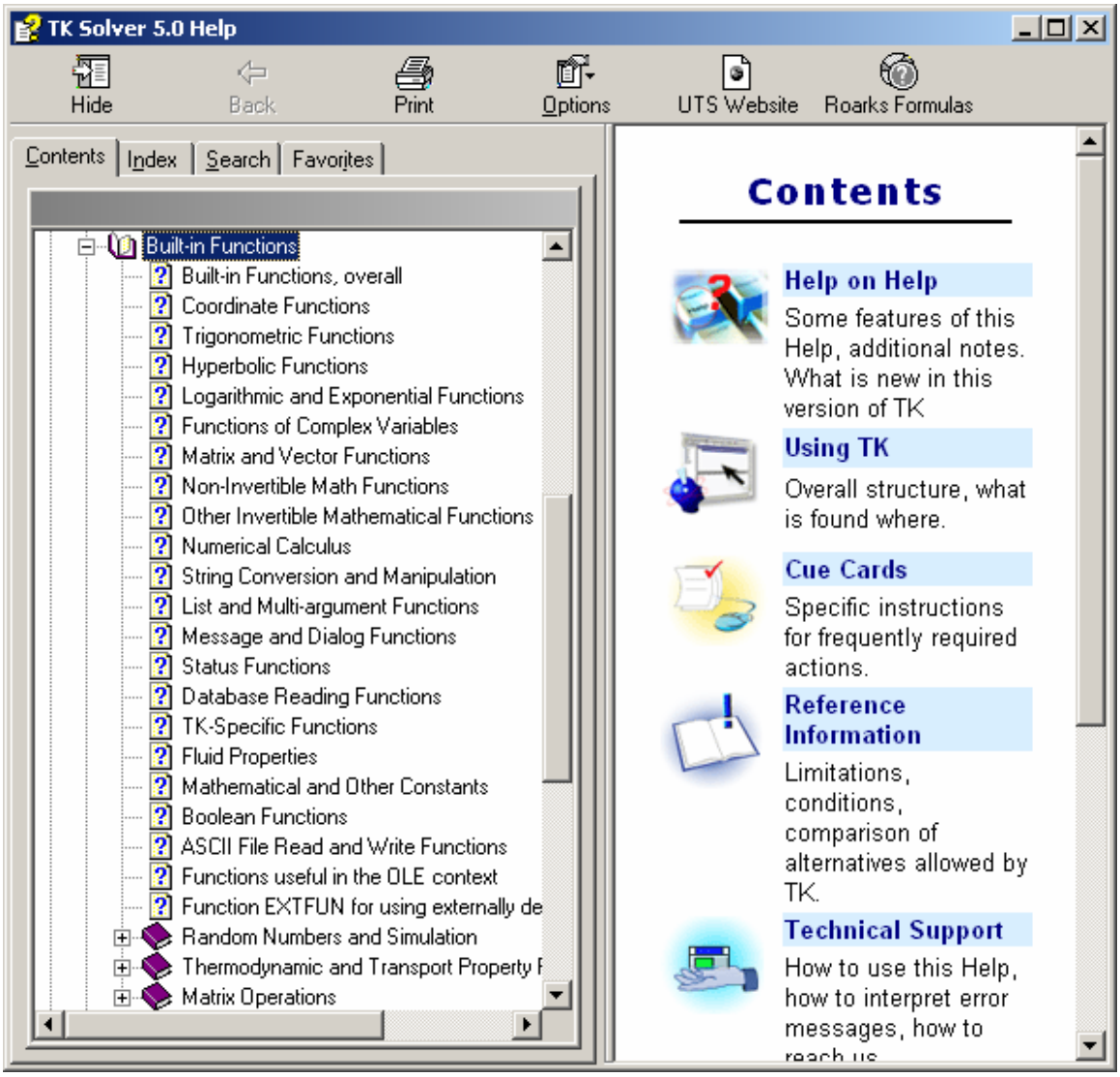
Functions

TK Functions can be classified as either built-in or user-defined.

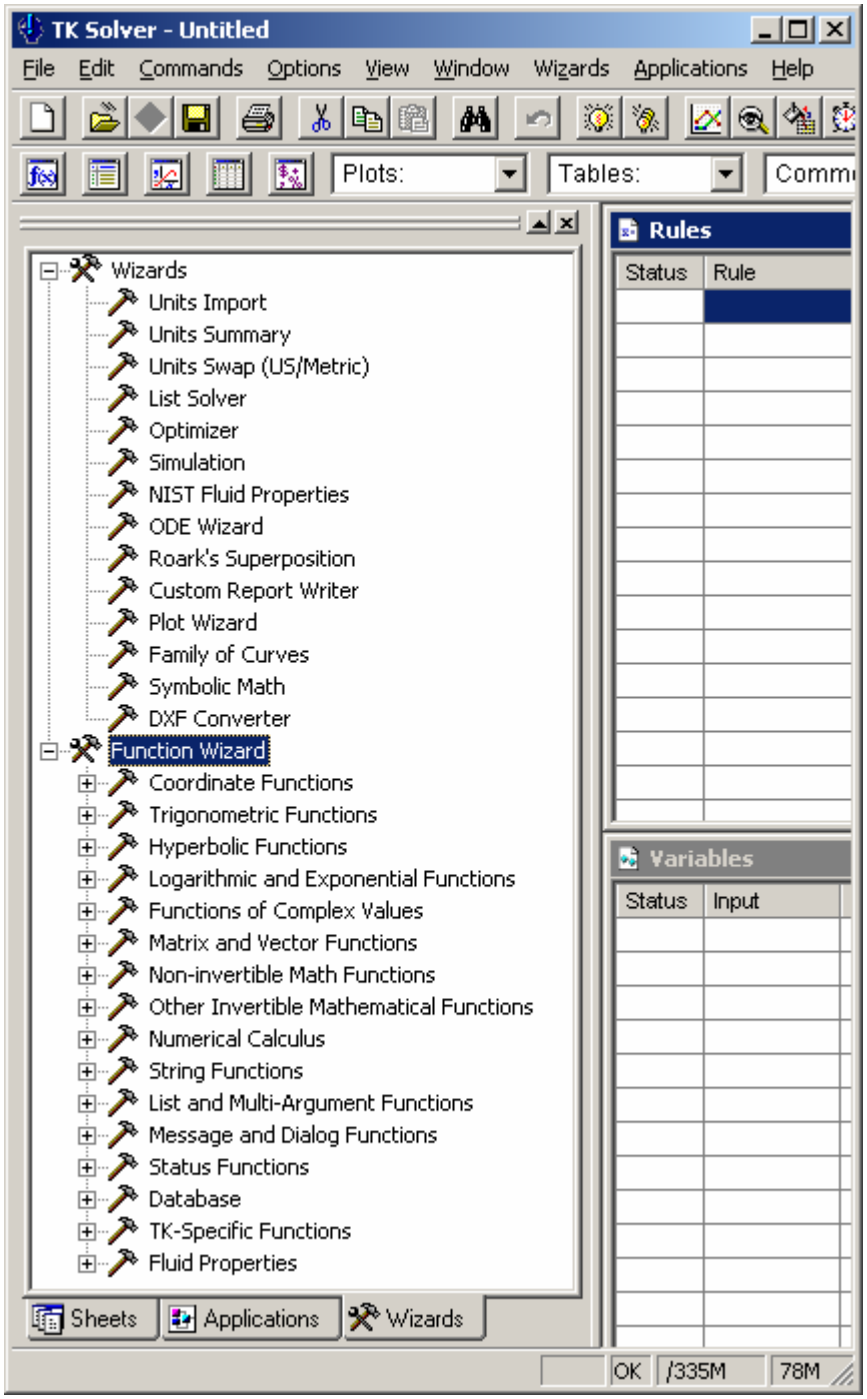
Listings of Available Built-in Functions

There are several ways to access listings of TK's built-in functions – the Help utility, the Wizards tab of the Navigation Bar, and the CTRL-F or CTRL-SPACE hot keys while editing equations.

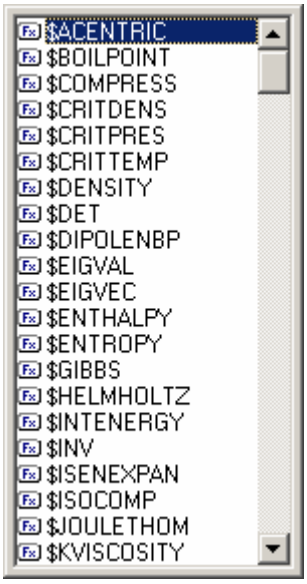
1. Reset TK and open the Help utility to the Table of Contents. Open the Functions and Wizards section and then select Built-in Functions. All the categories are displayed. Click on any group for further descriptions. Close the Help window when you are finished browsing it.



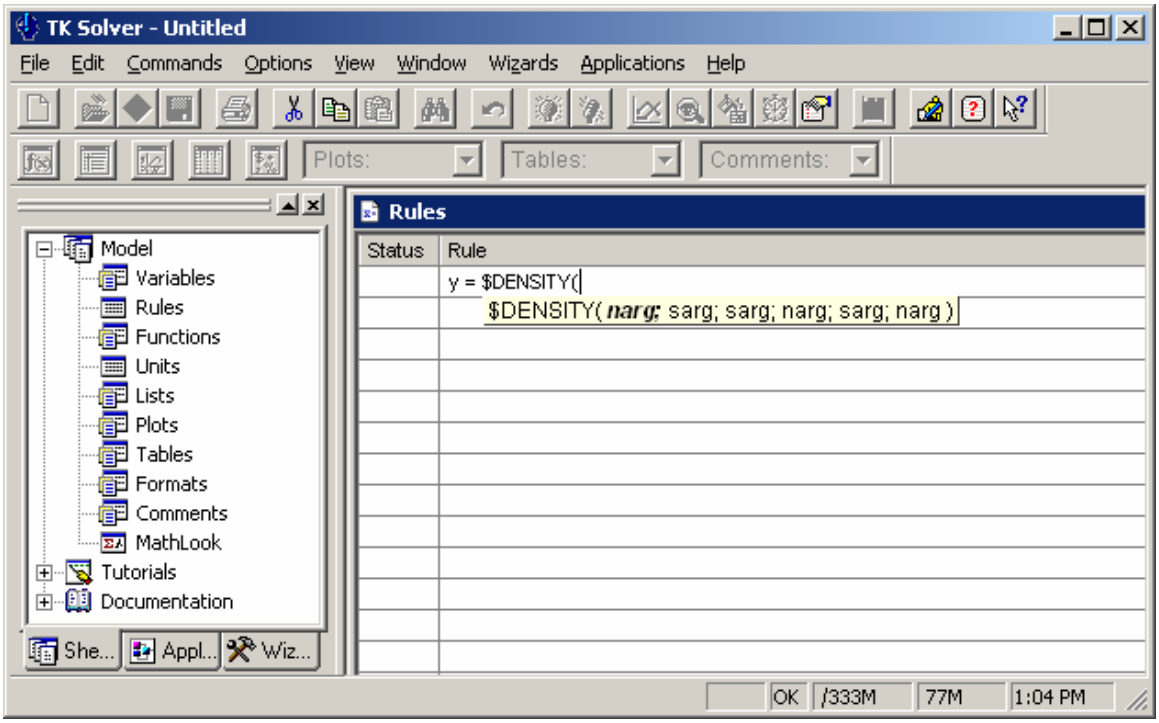
2. Click on the Wizards tab at the bottom of the Navigation Bar and then select Function Wizard for another listing of function categories. We will put these wizards to work later in this training session. When you are finished browsing this list, click the Sheets tab.



3. Begin typing “y = “ on the rule sheet and then press ctrl-f or ctrl-space and you will get an alphabetical listing of all the built-in functions. Note that those functions that start with special characters are at the top of the list.



When you select a function from the list, TK automatically enters it and displays a tool tip with a listing of the required value types. You will learn more about value types later. For now, press Escape to clear the rule.



TK Function Vocabulary

The words in italics below have special meanings with respect to TK.

1. Enter the equation y = Log(x). The *variable* x is the *argument* of the function and the variable y is the *result*. Because Log is an *invertible* function, TK's Direct Solver can solve for y given x or to solve for x given y. Log is a *built-in* function which means that it is always available whenever we are working in TK.

2. Switch to the Variable Sheet and enter 7, a *numeric value*, as the input for x and solve. The log function requires a numeric value as its argument and it returns a numeric value as its result.

3. TK variables can also have *symbolic values*. Enter the word red as the input for x. Note that TK automatically places an apostrophe in front of the word. Without the apostrophe, the expression would look just like a variable so the apostrophe is TK's way of differentiating between variable names and symbolic values. Symbolic values are sometimes called *strings or string values*. Symbolic values start with an apostrophe if there is just one word or are surrounded by double quotes if there are multiple words.

4. Solve again. TK reports an error, "Invalid Symbolic Computation". The log function cannot process a symbolic value.

5. Cursor down and enter red as the *name* of a third variable. Enter 10 as its input value. Now enter red as the input value of x again. TK replaces the word red with the value of the variable named red.

6. The rule y = Log(x) includes a *function reference*. Functions can also be *called*. Switch to the Rule Sheet and enter the rule call log(x;y) on the second line. The semicolon separates the argument variable from the result variable.

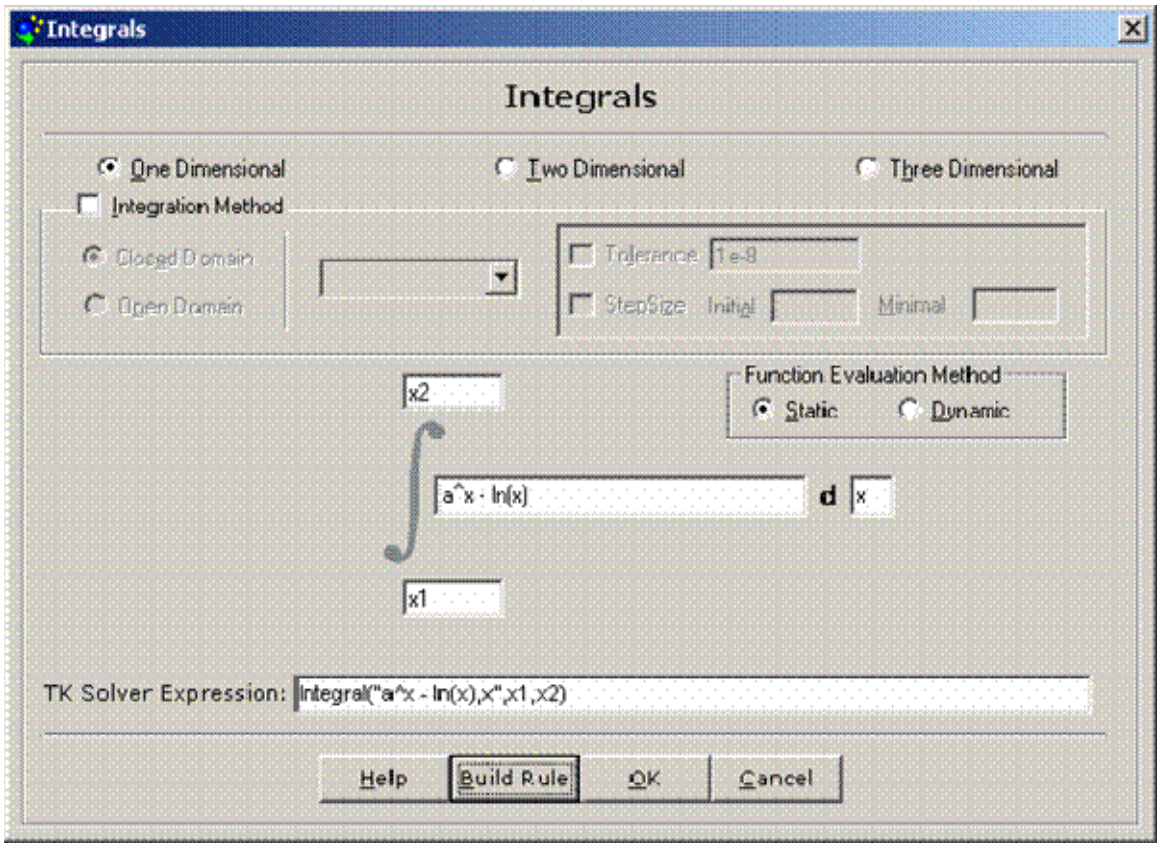
Function calls have the same effect as function references but they allow for 0 to 20 result variables whereas function references can only have 1 or 2 result variables. We will see the importance of function calls later. Solve to verify that the second rule is OK.

7. Click File New to reset the sheets and then enter the rule y = sum(x). Sum is also a built-in function. The function sum returns the sum of a list of numeric values. The argument variable x must have a symbolic value which refers to the name of a list.

8. Go to the List Sheet and create a list called abc. Open the subsheet for abc and enter the values 10, 20, 30, and 40. Go to the Variable Sheet and enter abc as the value for x. TK converts it to the symbolic value 'abc. Solve (F9) to see the result. Note that even though the rule processed a list of values, List Solving was not necessary. Remember that List Solving is the repeated process of rules on the rule sheet using lists of values associated with one or more inputs. In this example, we only need TK to process the rule sheet one time.

Function Wizards

The Wizards tab of the Navigation Bar presents a complete collection of wizards to help with using any of the TK built-in functions. You are presented with all the available options for each function and you fill in the blanks for everything you know about the inputs. There is a Help button which takes you directly to the online help related to the function. Once you have everything entered, click the Build Rule button and TK shows you what the expression will look like in TK. At that point, you can either click OK to paste the expression into the current field in TK or copy (CTRL-C) and paste (CTRL-V) the expression wherever you need it.



Rule Functions

If you have one or more rules that you intend to use repeatedly in one or more TK models you should consider the use of Rule Functions. Once created, Rule Functions can be saved and merged into other applications as needed. Rule functions can referenced repeatedly within other rules.

There's a nice example in the TK Library showing the efficiency of using rule functions. Go to the Engineering Examples section and study the Circuit Design model. The functions ser and par really simplify the rule sheet for that model.

Practice Problem:

TK Solver includes built-in functions for converting between rectangular and polar coordinates. The following example uses a rule function to extend this concept to allow for conversions to spherical coordinates.

1. Go to the Function Sheet and enter stord (short for spherical to rectangular) as the name of the function. Cursor to the Type column and enter R to define stord as a rule function then open the subsheet and enter the equations:

(x,y) = ptord(r,theta)
(z,r) = ptord(rho,phi)

The "d" suffix after the trig functions indicates that the calculation is done in degrees. The ptord function is a built-in function for converting between polar and rectangular coordinates with angles measured in degrees.

2. Enter rho,phi,theta in the Argument Variables field. Enter x,y,z in the Result Variables field. This maps the local variables with whatever expressions are used in referencing this rule function.

3. Activate the Rule Sheet and enter the following rule:

call stord(rho,phi,theta;x1-x0,y1-y0,z1-z0)

The variables and expressions in this rule are mapped to the local variables in the stord function according to their position. Those variables preceding the semicolon are mapped by position to the three Argument Variables in the function. Similarly, the three Result Variables in the function are mapped to the three expressions in the rule. For example, the expression x1-x0 is mapped to the local variable x.

4. Enter the following inputs on the variable sheet and solve.

rho=1; theta=-110; phi=26; x0=.5; y0=.7; z0=1

The stord function solves for the local variables x, y, and z and then TK computes x1, y1, and z1 on the Rule Sheet...just like a built-in function.

5. Go to the Rule Sheet and highlight the rule. Press CTRL-D and TK opens the stord function subsheet. Another way to quickly access a user-defined function is via the functions icon “f(x)” on the Object Bar. Once in the subsheet, use the Commands menu and select Examine or click the Examine icon (it looks like a magnifying glass) to invoke the Examine Command.

The Examine Command is used to evaluate an expression from anywhere in a TK model. It is especially useful for debugging functions by evaluating local variables. You can use it to check values, call functions, etc. Examine the value of the local variable r. That variable represents the projection of the distance rho onto the x,y plane.

6. Use the File Menu to Save the Window to a file called stordfun. The other sheets are not saved. The stord function might be useful in other models in the future and we will be able to merge it and use it as we would a built-in function.

Procedure Functions

Procedure functions (procedures) supplement TK's rule-based capabilities with sequential programming and processing. Procedures are called as subroutines from the rule sheet or from other functions. Procedures cannot call rules from the rule sheet. Procedures can not call on the Direct or Iterative Solvers to solve equations, however those solvers CAN repeatedly process procedures during an iterative process. Here is a complete listing of all the types of statements you can use in procedures.

An ASSIGNMENT statement has the form

variable_name = expression

or

list_element = expression

or

(name1, name2) = expression resulting in a pair of values

where name1 and name2 can be names of variables or list element identifiers or one of each. For example, 'XYZ[i] represents the ith element of the list XYZ. Assignment statements allow either = or := for the assignment operator. You may also call functions in procedure statements.

A GOTO statement has the form **goto label_name** .

A LABEL statement has the form **label_name:** .

A RETURN statement has the form **return** .

A FOR-NEXT loop construct has the form

FOR var_name = expression1 TO expression2 {STEP expression3}
{statements}
NEXT {var_name}

where expression1 and expression2 must evaluate to numeric values and the optional expression3 must evaluate to a non-zero numeric value (defaults to 1). The optional statements inside the loop can include other correctly nested for-next loops.

A CONTINUE statement has the form **CONTINUE {variable_name}** and takes control to the NEXT statement of the current loop, or to the NEXT statement of a loop at a higher level specified by the optional variable_name.

An EXIT statement has the form **EXIT {variable_name}** and takes control to the statement after the NEXT statement of the current loop, or to the statement after the NEXT statement of a loop at a higher level specified by the optional variable_name.

Practice Problem:

1. Reset all the sheets and open the Function Sheet. We will create a function which computes a factorial. Activate the Function Sheet and create a Procedure Function named fact.

2. Open the subsheet for fact and enter the information as shown below.

```
==================== PROCEDURE FUNCTION: fact ================
Comment:            Computes factorials
Parameter Variables:
Input Variables:      n
Output Variables:     z
S Statement-----------------------------------------------------------
 z = 1
 If OR(n=0,n=1) then return
 for i = 2 to n
  z = i*z
 next i
```

The Comment field helps us identify the contents of the function. There is one Input Variable and one Output Variable. These work in a similar fashion as Argument and Result Variables in Rule Functions. We use them to map values of local variables to the rest of the model.

3. Go to the Rule Sheet and enter c = fact(x). The two variables in this rule are mapped to the variables in the function – x is mapped to n and c is mapped to z.

4. Go to the Variable Sheet and enter 13 as the input for x and solve. Change x to 20 and solve again.

5. Enter an input value of 172 for x and solve again. An error! Open the fact subsheet and insert the following line in the first row to trap the error and provide a more helpful error message:

```
if n>170 then call errmsg("Function fact is not valid for n>170")
```

6. Suppose you also need the procedure to generate a list (called facts) of the factorials up to and including the nth. Insert the following as the first statement in the procedure.

```
Call blank('facts)
```

This will clear the facts list of any values from a previous run. Then insert the following statement just below the line z = 1 to place a 1 in the first element of the facts list.

```
'facts[1] = 1
```

Then insert the following statement just below the z = z*i statement.

‘facts[i] = z

This places the current value of z into the ith element of the facts list.

7. Suppose you would like to use different lists to store the factorials on different runs. You can make the list name a variable by removing the apostrophe in each of the three statements it is used. For example, ‘facts[i] becomes facts[i].

Then add facts to the list of input variables in the function.

n,facts

This tells the function that the value of the variable facts will be passed into the function. The name of the list will be an **input** to the function. The procedure will then place the values into that list.

8. Edit the rule on the rule sheet, adding a second input variable.

c = fact(x,facts)

9. Switch to the variable sheet and enter the name of the list you would like to use to store the factorials. When you solve (F9), TK generates the list on the list sheet. Change the inputs (x and facts) and solve again. A new value of c is displayed on the variable sheet and a new list is generated on the list sheet.

The TK Solver Library contains hundreds of procedures in various groupings. The Library contains both Tools and Examples. The examples show the tools in action. You can merge the tools into your own TK applications.

For example, there is a file in the Statistics section of the TK Library that computes factorials, combinations, permutations, and arrangements, for inputs of any size. The functions in that model utilize special arbitrary length integer calculation algorithms and represent large numbers as multiple list elements.

List Functions

List Functions allows us to create look-up tables with or without interpolation. List functions relate the elements of two lists. The mapping options are Table, Step, Linear, and Cubic, with the last two providing interpolation.

1. Start with a fresh set of TK Sheets. Go to the Function Sheet and create a List Function named f. Open the subsheet. List Functions require two lists of values and a mapping option. If the lists were previously defined in the TK model, the values will automatically appear within the function.

2. Enter color as the name of the Domain List and code as the name of the Range List.

```
==================== LIST FUNCTION: f ============================
Comment:
Domain List:        color
Mapping:            Table
Range List:         code
Element-- Domain------- Range---------------
1           'red              100
2           'white             200
3           'blue              300
```

3. Enter the values red, white, blue in the Domain column and 100, 200, 300 in the Range column.

4. Test the function using the Examine Command with the expression f('blue). TK should return the value 300. Close the Examine form.

5. Go to the Rule Sheet and enter the rule $y = f(x)$. The argument of the function, x, is mapped to the Domain and the result, y, is mapped to the Range. Switch to the Variable Sheet and enter 200 as the input for y. TK solves for x as the value 'white.

6. Open the function subsheet. Change the mapping option to Linear. Change the values in the color column to 1,2,3,4,5. Then change the values in the Range column to 2,4,7,23,87. TK will now interpolate within the given values.

7. Go to the Variable Sheet and enter 3.25 for x, blank y and solve. TK interpolates between 7 and 23 in the range list and provides the answer. Go back to the function subsheet, change the mapping option to Cubic and solve again.

8. Go to the Plot Sheet and create a line chart called f. Open the subsheet and enter color as the X-Axis List and code as the Y-Axis list twice -- one with Lines and one with Curves. When you plot (F7), you can see the difference between linear and cubic interpolation.

A Closer Look: Passing Values to and from Functions

We have already seen that variables in equations can be "mapped" with local variables in functions according to their position in the list of arguments and results for rule functions and inputs and outputs for procedure functions. List functions always have one input mapped to the domain list and one output mapped to the range list.

Values can also be passed to and from functions directly via list elements. For example,

Place('y,1) = temperature

This equation places the value of the variable temperature into the 1st element of the list y. That list element can be accessed from anywhere else in the TK model. So a rule function might include the rule

T = 'y[1]

List elements are accessible anywhere in TK and can be used to bypass the passing of variables as function arguments.

Parameter variables provide a means of passing values directly from the Variable Sheet to a function. This is useful for two reasons. First, since functions can call or reference other functions, you may need an efficient means of getting a value to a deeply nested function without having to pass the value through each of the intermediate functions. Second, some functions assume that another function will have a specific number of inputs as they process them. Additional values can be passed into the function using Parameter Variables.

Examples of built-in functions that reference other functions:

Numerical Integration:
INTEGRAL, INTEGRAL_2FIX, INTEGRAL_2VAR, INTEGRAL_3FIX, INTEGRAL_3VAR

Differential Equation Integrators:
ODE_RK4, ODE_RK5Adapt, ODE_BS, ODE_STIFFR, ODE_STIFFBS, ODE_RK5FINAL

TK Library tools also reference other functions. For example, various root finders repeatedly reference a function as they iterate to a solution. The Optimization section of the Library includes tools which repeatedly reference functions until a max or min is found. These tools pass inputs to the function but rely on parameter variables for passing constants from the Variable Sheet throughout the process.

Review Problem

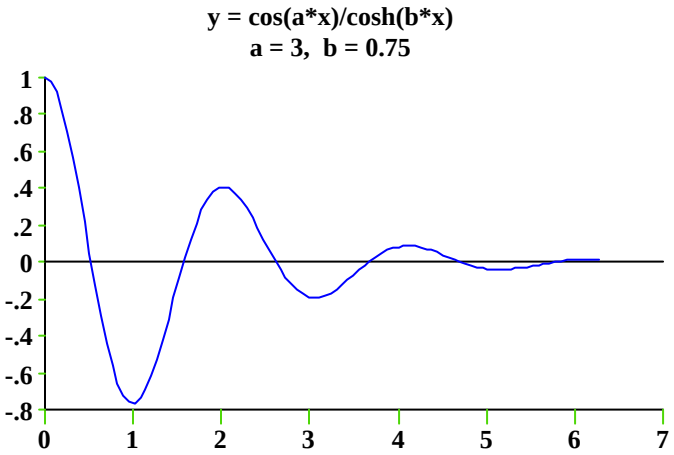
Just as we can find the square root of a variable or collate a list, we can also use TK to process functions. The TK Library has many tools for these tasks. The most commonly used tools are in the areas of differentiation, integration, optimization, and root finding. For example, we can differentiate a function with respect to its input variable. Likewise, we can find the input to a function which leads to result of zero, and so that input is a root of the function.

To further clarify, note that we do not differentiate variables or lists -- we differentiate functions. Given y = f(x), we do not integrate y from 0 to 1, we integrate f(x) as x ranges from 0 to 1. With this in mind, it is clear that we cannot integrate or differentiate a TK variable; we need to define a function and then process that.

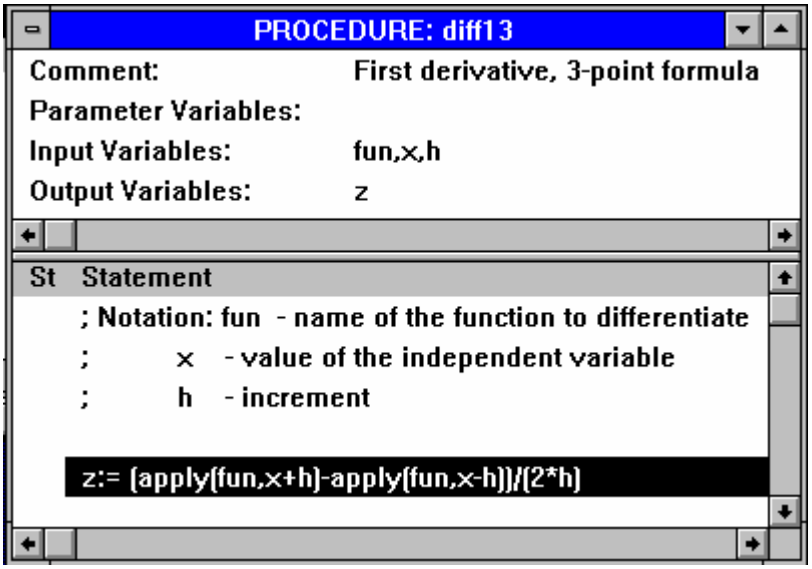
For example, suppose we have an equation such as **y = cos(a*x)/cosh(b*x)** and we are interested in studying the relationship between x and y as a and b remain constant.

- Our first step is to enter the equation on TK's Rule Sheet along with sample inputs on the Variable Sheet. Let's use a = 3 and b = 0.75. x will vary between 0 and 2*pi(). We'll list solve and generate a plot.

St	Input	Name	Output
L		y	-.76465998
L	1	x	
	3	a	
	.75	b	



- Now let's study the slope of the function at various values of x. To do this, we use the built-in differentiation tool from the TK Library. Select the tool called diff13 from the Mathematics section under DIFFINT and merge it into the current model.



The diff13 function requires 3 inputs and produces 1 output, as noted in the subsheet. The procedure features a single statement which evaluates the function at two points near the point of interest and then divides by the total interval, 2*h. The value of h should be relatively small, say 1E-3.

The built-in function APPLY is the key to this procedure. APPLY allows the name of the function to be a variable value. **APPLY('sqrt,9) = 3** and **APPLY('log,10) = 1**. So, if we have an expression **APPLY(fun,x)** , the value of the variable **fun** refers to the name of the function to apply to the value of **x** . In our model, we need to create a function for diff13 to process.

- The easiest way to do this is to copy the rule on the Rule Sheet and create a Rule Function on the Function Sheet. Call it **bounce** . Open the subsheet for bounce and paste the rule into the function.

  y = cos(a*x)/cosh(b*x)

- Enter **a,b** for the Parameter Variables, **x** for the Argument Variable, and **y** for the Result Variable.

  Parameter Variables: a,b
  Argument Variables: x
  Result Variables: y

Hmmm... Why are a and b listed as parameters and not arguments? The answer is that we want to differentiate the function with respect to x, while a and b remain constant. The diff13 procedure processes functions with one argument and one result (i.e. y = f(x)). Since the bounce function includes 4 variables, two of them must be passed in directly from the Variable Sheet as parameters.

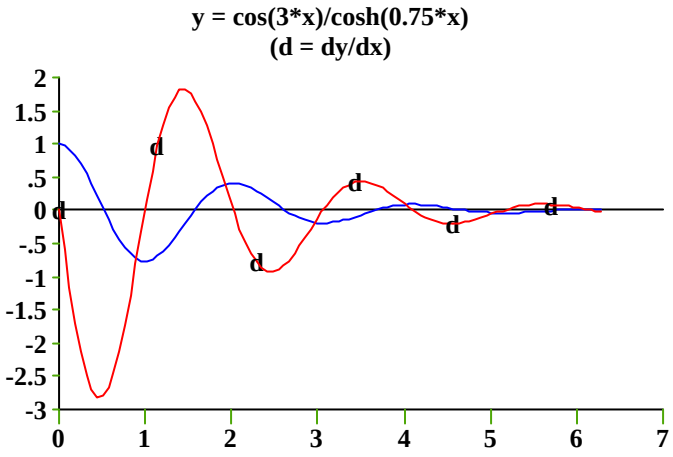
- Enter the reference to diff13 on the Rule Sheet and solve.

Rule
y = cos(a*x)/cosh(b*x)
slope = diff13('bounce,x,1E-3)

Input	Name	Output
	y	-.76465998
1	x	
3	a	
.75	b	
	slope	.03725461

So we see that the slope of the bounce function is .03725 at x = 1.

- For a plot of the derivative, change the status of **slope** to **L** and List Solve. Then add **slope** to the list of y-axis list names in the current plot.



- Let's see if we can find the value of x at which the bounce function is minimized. To do this, set the slope value to 0 and guess the value of x.

Input	Name	Output
	y	-.7647552
	x	.994893071
3	a	
.75	b	
0	slope	

Success. By guessing near the appropriate spots indicated by the plot, we could compute the locations of each of the local minimums and maximums in this fashion.