Review of Equation Solving, Iteration, and List Solving

- Enter the equations and inputs as shown below.

St	Rule
*	**a = exp(x) + x*y**
*	**b = sin(x*y) + x + y**

Sta	Input	Name	Output	Unit	Comment
	3.9	**a**			
		x			
		y			
	.75	**b**			

- Try to solve.

TK fails because both equations have two unknowns. Iteration is required.

- Declare x as a guess by typing a G in its status field.

Sta	Input	Name	Output	Unit	Comment
	3.9	**a**			
Gu	**0**	**x**			
		y			
	.75	**b**			

- Try to solve again.

After 10 iterations, TK displays the next guess it would like to make and stops.

Sta	Input	Name	Output	Unit	Comment
	3.9	**a**			
Gu	**.098936799**	**x**			
		y			
	.75	**b**			

- Solve again.

Success! TK placed the values in the Output column.

Sta	Input	Name	Output	Unit	Comment
	3.9	**a**			
		x	**1.46649847**		
		y	**-.29596539**		
	.75	**b**			

- Try a guess of 1 for x and solve again.

Same solution -- fewer iterations. The quality of the guess is important when solving nonlinear equations.

Rule of thumb: The default guess of 0 can be (and often should be) overwritten to avoid lengthy iterations or even errors (LOG(0), for example).

- Now, try a guess of 2 for x.

Sta	Input	Name	Output	Unit	Comment
	3.9	**a**			
		x	**2.03173803**		
		y	**-1.8345531**		
	.75	**b**			

A different solution! Hmmm... are there others? Let's investigate by introducing the concept of "error plots".

- Enter an INPUT (not a guess) of 1 for x and solve.

You'll get an error message and TK places error symbols next to the variables involved with the offending rule.

Sta	Input	Name	Output	Unit	Comment
	3.9	**a**			
>	**1**	**x**			
>		**y**	**1.18171817**		
>	**.75**	**b**			

St	Rule
	a = exp(x) + x*y
>	**b = sin(x*y) + x + y**

- Edit the second rule by adding an error term. Then solve again.

St	Rule
	a = exp(x) + x*y
*	**b = sin(x*y) + x + y + error**

Sta	Input	Name	Output	Unit	Comment
	3.9	**a**			
	1	**x**			
		y	**1.18171817**		
	.75	**b**			
		error	**-2.3569773**		

We see that the initial guess of 1 caused an inconsistency of -2.356... Our goal is to find values of x which make the value of error 0. We can List Solve over a range of x values

and store the resulting error values in a list. Then we can plot error vs. x and observe all the points at which the curve crosses the x-axis. These are the solutions.

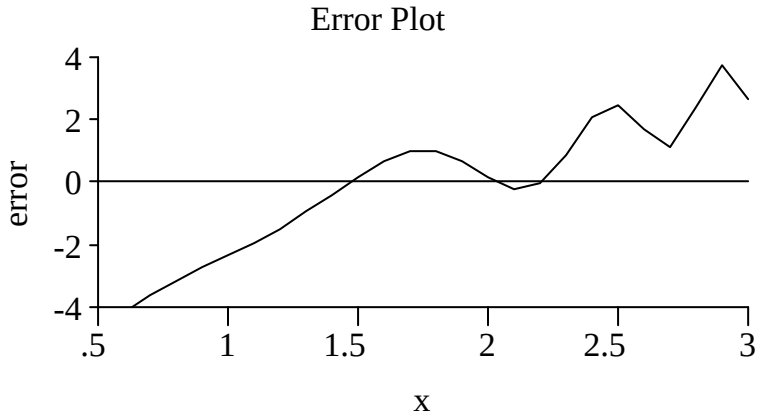
- Set up the Variable Sheet for List Solving (or use the List Solve Wizard)

Sta	Input	Name	Output	Unit	Comment
	3.9	**a**			
L	**1**	**x**			
		y	**1.18171817**		
	.75	**b**			
L		**error**	**-2.3569773**		

- Fill the x list with values ranging from 0.1 to 5.0 in steps of 0.1 and List Solve. Create a line chart with x on the x-axis and error on the y-axis. Be sure to set the Display Zero Axis field to X-Axis.

The plot appears to cross the x-axis 3 times. Let's zoom in for a better look.

- Click and drag on the plot to zoom to the region shown below.



This gives a clearer picture. We've already seen that given a guess of 1 or 2, TK finds the first two roots. Now, let's see if TK can iterate to the third root. It's not easy...

- INPUT 0 for error, GUESS 3 for x, and solve.

Sta	Input	Name	Output	Unit	Comment
	3.9	**a**			
L		**x**	**2.03173803**		
		y	**-1.8345531**		
	.75	**b**			
L	**0**	**error**			

Rats! It returns the second root. We may have to give a fairly close guess.

- Try a guess of 2.25 for x and solve again.

Sta	Input	Name	Output	Unit	Comment
	3.9	**a**			
L		**x**	**2.20618144**		
		y	**-2.3483897**		
	.75	**b**			
L	**0**	**error**			

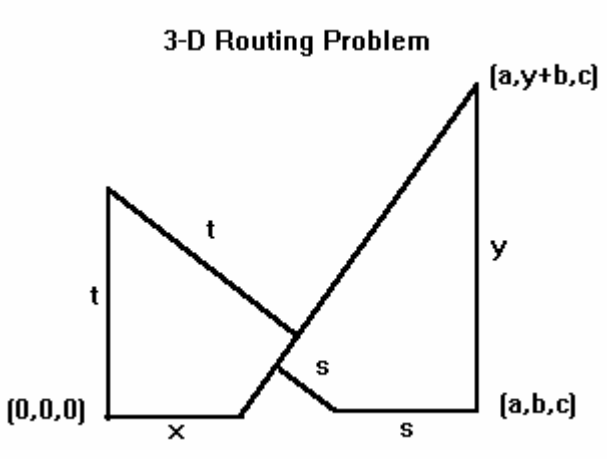
Success! You should now have a better understanding of how TK iterates to solutions.

Rule of Thumb: If you are concerned about the potential for multiple solutions to a problem involving a single guess variable, use error plots to get a better picture.

Supplying Educated Guesses to TK

TK relies on a modified Newton-Raphson method for iterative solutions. As you saw in the previous section, your choice of guesses can affect the solution. This exercise shows you ways of controlling the solution process.

Consider the following waveguide routing problem.



Without going into all the engineering details, here's what's going on. Several lines extend from a single unit to identical amplifiers. Each line must be accurately the same length to ensure that the signals are in phase. This involves the solution of the following simultaneous nonlinear equations.

St	Rule
*	(c^2 + (y+b)^2)/((a-x)^2 + (y+b)^2 + c^2) = ((2*x*t)/(x^2+t^2))^2
*	(c^2 + (a-x)^2)/((a-x)^2 + (y+b)^2 + c^2) = ((2*y*s)/(y^2+s^2))^2

- Enter these and copy them into MathLook to verify placement of parens, ratios, etc.

In MathLook form...

$$\frac{c^2 + (y+b)^2}{(a-x)^2 + (y+b)^2 + c^2} = \left[\frac{2 \bullet x \bullet t}{x^2 + t^2}\right]^2$$

$$\frac{c^2 + (a-x)^2}{(a-x)^2 + (y+b)^2 + c^2} = \left[\frac{2 \bullet y \bullet s}{y^2 + s^2}\right]^2$$

The variables

a,b, and c represent the x, y, and z coordinates and s, t, x, and y are the distances. In most cases, we know the values of a, b, c, s, and t and need to know the values of x and y. We also require that y>s and 0<x<t .

We know that iteration will be necessary. Both unknowns appear more than once in each equation. In fact, we will need to supply guesses for both x and y.

Our goal is to create a TK Solver model which reliably solves for x and y without the user having to supply guesses.

- Reorganize the Variable Sheet and enter the following sample inputs:

Sta	Input	Name	Output	Unit	Comment
	10	**a**			
	5	**b**			
	7	**c**			
	2	**s**			
	3	**t**			
		x			
		y			

- Try various guess values for x and y. Do you always get the same solutions? There's gotta be a better way...

- Try guessing 2*s for y, and t/2 for x.

Sta	Input	Name	Output	Unit	Comment
	10	**a**			
	5	**b**			
	7	**c**			
	2	**s**			
	3	**t**			
		x	**1.52809297**		
		y	**4.31788045**		

Hmmm... this works but it's a little tedious. Every time the inputs change, we have to change the guess entries. How can we automate this?

- Add the following equations to the Rule Sheet.

St	Rule
*	**x = xg*t/2**
*	**y = yg*2*s**

The new variables appear on the Variable Sheet. What if xg and yg are always initially guessed as 1? That will make x = t/2 and y = 2*s, which is what we want!

- Open the subsheet for the variable xg and enter 1 as the value for the First Guess.

Status:	
First Guess:	**1**
Associated List:	

- Close the subsheet and make the same entry in the variable yg subsheet.

- Solve (without any guesses on the Variable Sheet) and TK automatically triggers the first guesses for xg and yg and converges in a few iterations.

Sta	Input	Name	Output	Unit	Comment
	10	**a**			
	5	**b**			
	7	**c**			
	2	**s**			
	3	**t**			
		x	**1.52809297**		
		y	**4.31788045**		
		xg	**1.01872864**		
		yg	**1.07947011**		

Note that the final values xg and yg reflect the relative accuracy of the "guess equations" on the Rule Sheet. The closer to 1 the better.

- Try solving with the new set of inputs shown below.

Sta	Input	Name	Output	Unit	Comment
	12	**a**			
	6	**b**			
	4	**c**			
	4	**s**			
	1	**t**			
		x	**.599102373**		
		y	**14.930068**		
		xg	**1.19820475**		
		yg	**1.8662585**		

It works.

List Solving and Iteration

During iteration, a poor guess may force TK to run into an error condition such as a square root or log of a negative number. List solving increases the odds of such an error. Fortunately, there are ways of turning the odds back in your favor.

Brief Overview

List Solving is the repeated processing of the rules, using a list of input values for one or more variables. Any output variables can be associated with lists to store the resulting values. Before list solving, it's always a good idea to make sure the model solves once for a sample set of inputs.

If iteration is required to get solutions, there are two ways of providing guesses:

1. Provide a first guess on a variable subsheet. (variable status Output)
2. Provide a list of guesses. (variable status ListGuess)

VERY IMPORTANT NOTE: A variable assigned the status G (without the L) ***will not*** act as a guess throughout list solving. The guess will be used only for the first element. From that moment on, the variable's status is Output only. So, when list solving, output variable status should either be L, LG, or O -- never G!

In many cases, a default first guess works for the entire range of values. It's definitely the easiest method. For those cases where more control is required in supply guesses, there are several ways of generating a list of guesses:

1. If we expect the solutions to be steadily increasing or decreasing, we can solve for the first and last elements and then use the List Fill command to complete the list using either linear or log spacing.
2. We can program TK to use the current solution (or some function of it) as the next guess.

Let's work an example to see how this works. The equations below are related to aircraft engine bleed systems. These systems are used to power other operations on the aircraft.. In systems involving two or more engines, the interrelationships between the engines must be accounted for to determine overall flow rates.

- Enter these equations and copy the second one into MathLook to verify your typing.

St	Rule
	wd = w1 + w2
	w2 = w1*sqrt((m*p1 + dp*w2 - ps)/(p1 + dp*w1 - ps))
	dw = w2 - w1

$$w2 = w1 \bullet \sqrt{\frac{m \bullet p1 + dp \bullet w2 - ps}{p1 + dp \bullet w1 - ps}}$$

- Set up the Variable Sheet as shown below, including the input values. (Skip the comments.) Enter a guess of 1 for w1 and solve.

Sta	Input	Name	Output	Unit	Comment
					Fluid Systems Analysis
	5	**wd**			**total mass flow**
		w1	**2.37122348**		**mass flow, engine 1**
		w2	**2.62877652**		**mass flow, engine 2**
		dw	**.25755304**		**difference in mass flow**
	1.04	**m**			**pressure imbalance ratio**
	40	**p1**			**engine 1 port pressure**
	-.3	**dp**			**bleed valve pressure drop**
	32.64	**ps**			**static pressure**

That was easy enough...

- Open the subsheet for w1 and enter 1 as the First Guess value. Close the subsheet and solve again to test that it works.

Our goal is to vary wd from 0 to 100 in steps of 2 and solve for w1, w2, and dw.

- Set up the Variable Sheet, fill the input list, and List Solve.

After a dozen or so elements are complete, TK starts beeping at us! (Ctrl-Break stops the beeping.)

- Activate the Options Setting for Stop on List Error and list solve again. This time TK stops as soon as it detects an error.

The status of the offending rule reads, **`> SQRT: Argument error.`**

- Create a table with the four lists and observe the results to this point.

Element	wd	w1	w2	dw
15	**28**	**13.073**	**14.927**	**1.8535**
16	**30**	**13.982**	**16.018**	**2.0362**
17	**32**	**1.0002**		
18	**34**	**1**		

During the solution process at element 17, TK tried to take the square root of a negative number. The guess of 1 for w1 doesn't seem to work when wd = 32. It forces the numerator and denominator of the ratio in the second rule to have different signs, which causes an error in the sqrt function. Since the default first guess strategy did not work, we need to supply a list of guesses. (Note that we might also try building in an educated guess as detailed in the last section but that technique is usually only necessary for application development. In this case, we just want a table of solutions.)

Based on the solutions in the table, we might conclude that w1 increases as wd increases. It also makes sense intuitively. We might try to find the solution for wd = 100 and then fill the list with evenly spaced values.

- Go to the Variable Sheet and try to solve for w1 given wd = 100. It's not easy. Most guesses cause error conditions. Fortunately, we know some limits for w1 -- it must be somewhere between 0 and wd and the solutions in the table have been less than half of wd. So w1 should be somewhere between 0 and 50. (Try guessing 30...)

Sta	Input	Name	Output	Unit	Comment
L	**100**	**wd**			**total mass flow**
L		**w1**	**34.2999083**		**mass flow, engine 1**

- Now place the solution for w1 into the last (51st) element of the list using the Put Values to Lists Command.

- Next, fill the 51 values of the w1 list using the Fill by Range Option with Linear Spacing from 0 to w1.

- Change the status for the variable w1 to LG and List Solve.

This time the model fails at wd = 46. Apparently the relationship between wd and w1 is not approximately linear. We need to provide better guesses.

Element	wd	w1	w2	dw
22	**42**	**19.319**	**22.681**	**3.3612**
23	**44**	**20.185**	**23.815**	**3.6293**
24	**46**	**15.778**		

Guess Extrapolation

An efficient way to overcome our guessing problem is to use previous solutions to make the current guess.

- Add the following rule to the Rule Sheet:

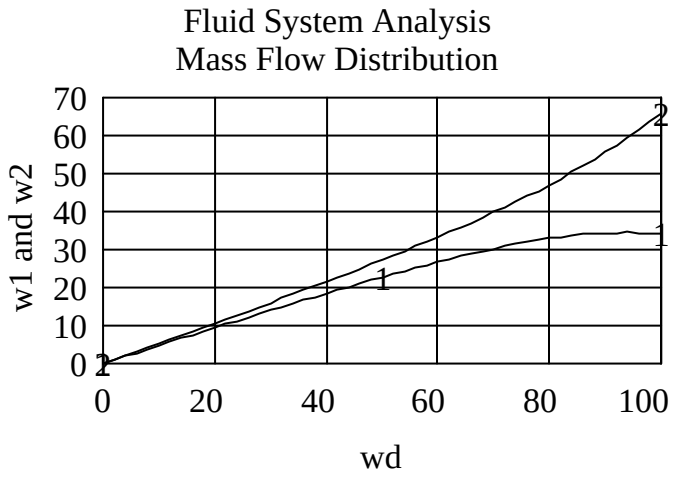
St	Rule
*	**place('w1,elt()+1) = 2*w1 - elt('w1,elt()-1,w1)**

This rule adds the difference between the current and previous solution to the current solution and places the value in the next element of the guess list. The built-in elt function serves two purposes. First, elt() identifies the current element number during list solving. Second, it is used to get a value, if it exists, from a list. If the value does not exist (there is no 0th element in TK), the rule uses w1.

This rule assumes that you have set up the Variable Sheet to access a list of guesses -- that is, the status of w1 must be LG. There must be at least 1 value in the list before starting the List Solver. In this case, we have half a table of values, so we can continue.

- List Solve again. (It's gonna' work this time!)

Here's the resulting plot...



...and here's part of the table.

Element	wd	w1	w2	dw
46	**90**	**34.365**	**55.635**	**21.27**
47	**92**	**34.457**	**57.543**	**23.085**
48	**94**	**34.491**	**59.509**	**25.018**
49	**96**	**34.471**	**61.529**	**27.057**
50	**98**	**34.405**	**63.595**	**29.19**
51	**100**	**34.3**	**65.7**	**31.4**

One final tip/trick to keep in mind:

If guess extrapolation fails, try using smaller increments for your input list.

Time-Step Solutions: Tank Draining

This example is taken from an excellent book by Johannes Gessler of Colorado State University, entitled TK Solver -- A Tutorial, published by McGraw-Hill. The book features many enlightening exercises and we encourage anyone wishing to hone their TK skills to purchase a copy.

Water flows from a tank through a pipe and discharges into air at its end. How long does it take for the water level (H) to go from 50 ft to 20 ft?

In effect, we are solving the differential equation q*dt = a*dH , where **a** is the horizontal area of the tank, **dH** is the change in the water level, **q** is the flow rate, and **dt** is the increment of time. We also know:

1. The friction factor equation is f = (-2 * log(e/(3.7*d) + 2.51*nu/(v*d*sqrt(f))))^-2
2. The head loss equation is H = f*L/d*v^2/(2*g)
3. The flow rate equation is q = .25*v*pi()*d^2
4. The drain pipe has length (L) 2000 feet and diameter (d) of 8 inches (2/3 ft)
5. The acceleration due to gravity (g) is 32.2 ft/s^2
6. The absolute roughness of the pipe (e) is .00085 ft
7. The kinematic viscosity (nu) is .00001 ft^2/s
8. The horizontal area (a) is 500 ft^2.

Solution:

- Enter the three equations above for f, H, and q on the TK Rule Sheet.

- Rearrange the Variable Sheet as shown below and enter inputs for the known values as described above. (An equation involving the variable **a** will be entered later. Ignore it for now.)

Since **f** is an unknown appearing twice in the first equation, a guess is necessary to solve for it.

- Open the subsheet for **f** and enter a default first guess of .01 (f is typically between .01 and .03). Then, close the subsheet and solve.

Sta	Input	Name	Output	Unit	Comment
	50	**H**		**ft**	**water level in the tank**
		f	**.021384747**		**friction factor (default guess set to .01)**
	2000	**L**		**ft**	**length of pipe**
	.666666667	**d**		**ft**	**diameter of pipe**
		v	**7.08**	**ft/s**	**average velocity in pipe**
	32.2	**g**		**ft/s^2**	**gravity**
	.00085	**e**		**ft**	**absolute roughness**
	.00001	**nu**		**ft^2/s**	**kinematic viscocity**
		q	**2.47**	**ft^3/s**	**flow rate**

At a given water level, we are able to compute the corresponding flow information.

- Add the following equations to the Rule Sheet. These represent the differential equation to be solved.

    i = elt()
    if i <= 1 then dt = 0 else (q+'q[i-1])/2*dt = a*('H[i-1]-H) ; change in volume

The first rule uses the built-in ELT function to return the current time step number during solving or list solving. ELT = 0 during direct solving and returns the current element number during list solving.

The second rule solves for the time change, initially setting it to 0, with subsequent computations done by dividing the changes in volume by the *average* flow rates during each of the time steps.

Note that without the if condition, TK would fail during the first element of list solving due to a reference to a 0th element of a list.

- Add an equation to sum the individual time changes for the total draining time.

    if i <= 1 then place('total,1) = 0 else place('total,i) = 'total[i-1] + dt

Again, the conditional expression is used to avoid an error during the initial time step. The built-in place function is used to send the total time value at each step to a list.

- Set up the Variable Sheet for list solving. **H**, **v**, **q**, and **dt** are associated with lists and **H** is the input. The variable **a** now appears at the bottom of the Variable Sheet. Input the value 500 for **a**.

Sta	Input	Name	Output	Unit	Comment
L	50	H		ft	water level in the tank
		f	.021384747		friction factor (default guess set to .01)
	2000	L		ft	length of pipe
	.666666667	d		ft	diameter of pipe
L		v	7.08	ft/s	average velocity in pipe
	32.2	g		ft/s^2	gravity
	.00085	e		ft	absolute roughness
	.00001	nu		ft^2/s	kinematic viscocity
L		q	2.47	ft^3/s	flow rate
L		dt	0	s	time step size
	500	a		ft^2	horizontal area of tank

- Using the List Fill command, fill the H list with values from 50 to 20 in steps of -1.

- List Solve and create a formatted summary table of the results.

Element	H	v	q	dt	total
1	50	7.08	2.47	0	0
2	49	7.01	2.45	203.22	203.22
3	48	6.94	2.42	205.33	408.55
4	47	6.87	2.4	207.5	616.05
5	46	6.79	2.37	209.75	825.8
6	45	6.72	2.34	212.07	1037.88
7	44	6.64	2.32	214.47	1252.35
8	43	6.56	2.29	216.96	1469.31
9	42	6.49	2.26	219.53	1688.83
10	41	6.41	2.24	222.19	1911.03
11	40	6.33	2.21	224.95	2135.98
12	39	6.25	2.18	227.82	2363.8
13	38	6.17	2.15	230.8	2594.61
14	37	6.08	2.12	233.9	2828.51
15	36	6	2.09	237.13	3065.64
16	35	5.91	2.06	240.49	3306.13
17	34	5.83	2.03	244	3550.14
18	33	5.74	2	247.67	3797.81
19	32	5.65	1.97	251.51	4049.32
20	31	5.56	1.94	255.53	4304.85
21	30	5.47	1.91	259.75	4564.6
22	29	5.38	1.88	264.18	4828.79
23	28	5.28	1.84	268.85	5097.64
24	27	5.18	1.81	273.77	5371.41
25	26	5.09	1.78	278.97	5650.39
26	25	4.98	1.74	284.48	5934.87
27	24	4.88	1.7	290.33	6225.2
28	23	4.78	1.67	296.55	6521.74
29	22	4.67	1.63	303.18	6824.92
30	21	4.56	1.59	310.28	7135.2
31	20	4.45	1.55	317.89	7453.09

Using 1 inch step sizes for H, the resulting drain time total is computed as 7453.09. How accurate is this and what are the effects of different step sizes? The worst estimate we could get would be to have a single step of 30 inches, in which case dt is computed as 500*30/2.0131645 = 7450.96. 1000 steps results in 7453.09. We should feel confident.

Families of Solutions/Curves

You can use TK to study the effects of two variables on another by generating families of solutions. The result is a family of curves.

One way to generate families of curves is to repeatedly list solve over the same range of values for one list, substituting a new constant value and a new list name each time. This strategy is the one to use when iteration is required.

Another way to get the job done is to use a procedure function. Let's try an example.

Suppose you have the following equation and you want to see what happens to z as a varies from 1 to 2 and as t varies from -2 to 2.

f = t^4 + a*t^3 + a^3*t^2 + a^3*t + a^4

- Create a procedure function called f, open the subsheet and enter the following statements.

St	Statement
	a = 1
	for i =1 to 11
	t = -2
	for j = 1 to 41
	't[j] = t
	'f[i][j] = t^4 + a*t^3 + a^3*t^2 + a^3*t + a^4
	t = t + .1
	next j
	a = a + .1
	next i

The function has no Input or Output Variables. The lists will be generated automatically.

- Use the Examine Command to call the function -- call f()

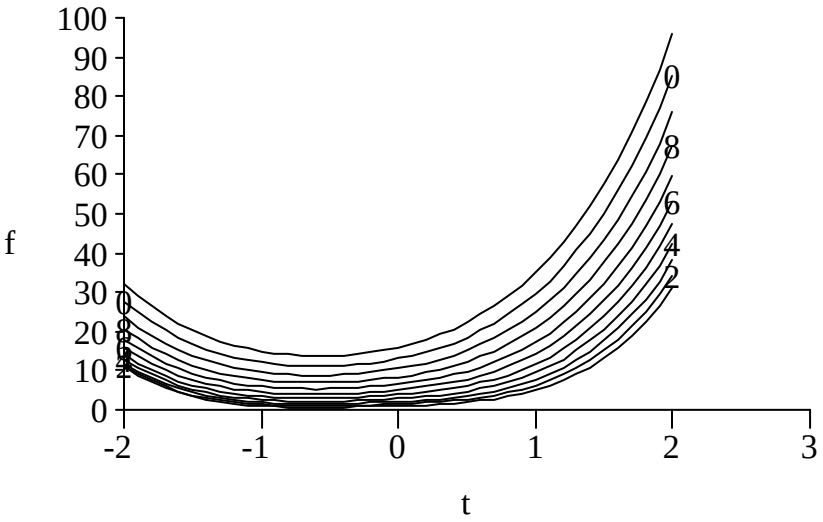
- Open the List Sheet and see that TK created 13 lists.

The list f contains the names of the lists in the family. It is also referred to as the master list. The other 12 lists contain results for each of the 11 values of a. TK automatically names the lists f#1, f#2, etc. The solutions for a=1 are in the list f#1; a=2 solutions are f#2 and so forth. The t list was generated and overwritten 11 times -- not efficient programming but the outcome was alright.

- Copy the 11 f data list names and create a Line Chart called f. Open the subsheet and paste the list names in as the Y-Axis lists. Enter appropriate symbols and counts.

Enter t as the name of the X-Axis List and display the plot.

f = t^4 + a*t^3 + a^3*t^2 + a^3*t + a^4
a = 1, 1.1, 1.2,...



Data Types and Import/Export Methods

There are two types of data in TK Solver -- individual values and lists.

Individual values are not easily imported or exported to or from other applications. They can be manipulated using Visual Basic through OLE. The TK Users Guide provides details.

Lists are much easier to access and manipulate. Lists are always stored on TK's List Sheet. They can be stored separately from the rest of the model using the File Export Command. They can also be exported during model execution using the built-in function ASCIIWRITE.

File Export, ASCII file: Save all or a highlighted range of lists to an ASCII file. The names of the lists will be saved. The units will not be saved.

File Export, DIF file: Same as ASCII storage, except that the units are saved. This is really only useful for subsequent loading into other TK models, since most other programs do not read DIF format.

File Export, WKS file: This stores the data in a format readible by most spreadsheets.

ASCIIWRITE function: Copies lists, without units, to files during model execution.

File Export, text file: Do not use this for storing lists. This option should only be used for storing TK Comment Sheets.

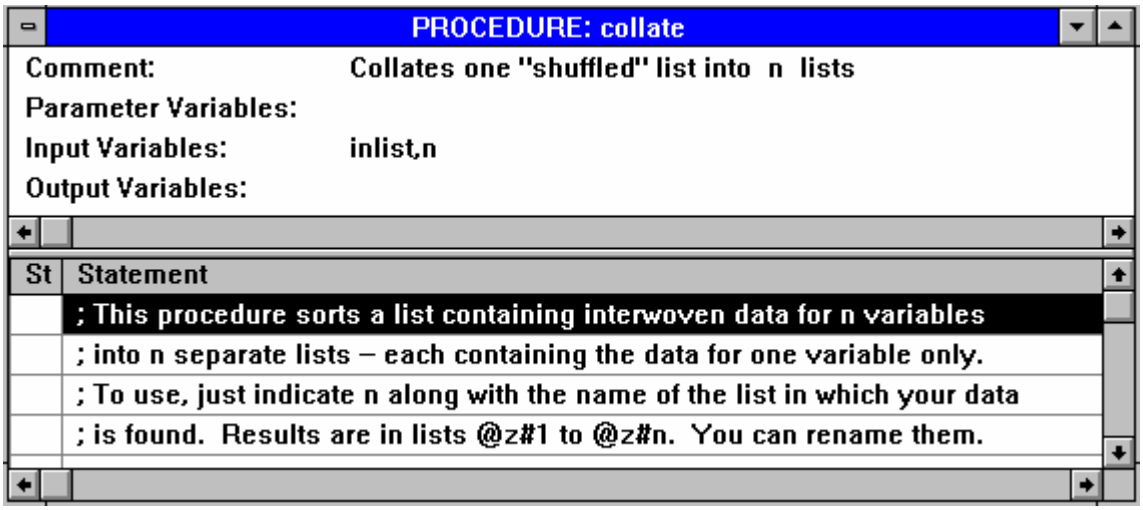
If you have data stored in a text file, you can import it as an ASCII file but it will load as a single list. Here's an example. Suppose you have three columns of data in a text file and you'd like to use the data in a TK model. The first column represents age, the second height, and the third weight.

24	73	172
33	67	191
51	76	236
29	74	220
38	71	161
48	69	166
41	71	189

If you import this file into TK as a text file, the data will appear on the Comment Sheet. Unfortunately, you can't do much with comments except display them. You can't calculate with them. We need the data on the List Sheet. So, we import the file as an ASCII file. What we get on the List Sheet is a single list with no name. We know it's there because it has 21 elements. We give it a name, say, data, and open the subsheet. The data has been mixed together.

24
73
172
33
67
191
...

We need a tool to collate the lists for us. The TK Library offers such a tool, in the LISTUTIL section. Merge the collate tool and use the Examine Command to call it.



The function requires two inputs -- the name of the list to collate and the number of output lists. **Call collate('data,3)** does the job. The List Sheet will now have four new lists representing the output matrix.

@z
@z#1
@z#2
@z#3

The first list, @z, holds the names of the columns. The other three hold the data. You can rename them as you like, place them in a table, plot, etc.

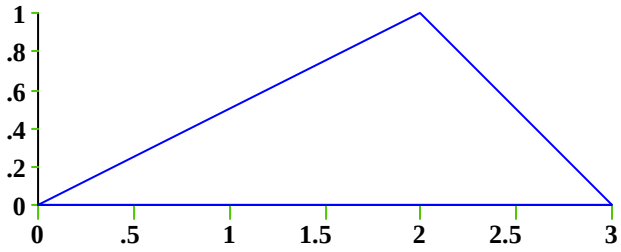
Lists and Plots

TK Solver's plotting capabilities are based on connecting pairs of x,y coordinates. If two consecutive elements of both the x and y axis lists have numeric values, TK connects the points and forms a line.

- Let's try it. Create a table with two lists, x and y, as shown below.

Element	x	y
1	0	0
2	2	1
3	3	0
4	0	0

- Next, create a line chart of the data with isotropic scaling.



Hmmm... four points make a triangle. Now what if we wanted to add a circle to this plot, centered at (2,1) with radius 1? We could program our own procedure function to do this or use tools from the TK Library. In the Utilities section, under GRAPH is a tool called pickit.

- Merge the pickit tool into your model.

The Function Sheet is packed with goodies. Let's try the circle function.

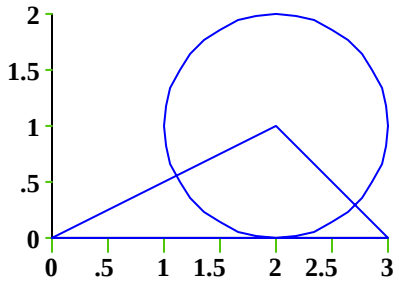
- Click on the function subsheet icon and select circle.

The function requires 7 inputs **x,y,x0,y0,r,phia,phiz** representing the two list names, the center coordinates, the radius, and the starting and stopping angle. We can use the Examine Command to call the function.

- Call circle('x,'y,2,1,1,0,360)

TK processes the procedure and appends the the new x,y coordinates to the table and plot.

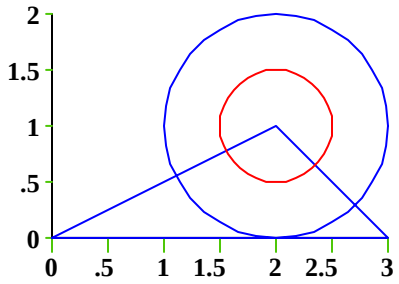
Element	x	y
1	0	0
2	2	1
3	3	0
4	0	0
5		
6	3	1
7	2.98480775	1.17364818
8	2.93969262	1.34202014
9	2.8660254	1.5



The blank row in the table breaks the connection between the sets of coordinates. The function was set up to produce the blank line automatically.

How do we add shapes with different colors? Use different y-axis lists.

- Add another circle at the same location with a radius of only 0.5 instead of 1. Specify ‘x and ‘v as the coordinate lists. Add v to the set of y axis lists on the plot subsheet.



(The inner circle should be displayed in red on your monitor.)

The other pickit functions work similarly, as do the other list processing functions in the TK Library.

Random Number Generation and Simulation

TK Solver can be used to repeatedly process rules over ranges of inputs with the result being tables and plots of solutions. These results show the total range of possible solutions. Suppose we are interested in studying the likelihood of observing certain results. We can design for what's most likely (say, 99.5% of the time) instead of what's theoretically possible. We can do this using simulation in TK Solver. Instead of list solving over the range of possible inputs, we list solve through random numbers from an assumed distribution.

How do we generate random numbers in TK? The Statistics section of the TK Library features a variety of functions for this.

- Using the Menuing System, load the file Random from the DISTAT area of the Statistics section of the Library.

As the comment reports, the random number generators all work in a similar fashion. They are listed in the on-screen comment and defined on the Function Sheet.

Let's try an example. A certain baseball player is assumed to have a batting average of .282, meaning he got a hit in 28.2% of his at bats. Assuming 4 at bats per game, simulate 1000 games, recording the number of hits in each game.

This example requires the use of a binomial distribution and the random number generator is called ranbin. According to the ranbin function, we must supply a starting seed, the number of values we wish to generate, the probability of getting a hit on any given at bat, the number of at bats per game, and the name of the list in which to store the results.

- Using the Examine Command, enter **call ranbin(13579,1000,.282,4,'hits)**

TK generates a list called hits on the List Sheet. Now let's see if we can answer a few questions about the 1000 games we just simulated.

1. What was his batting average during the 1000 simulated games?
To answer this question, we simply sum the list to get the total number of hits and divide by 4000, the total number of at bats. In this case, the answer is .281 (we expected it to be close to .282).

2. What was the resulting distribution of hits? How many times did he get 0, 1, 2, 3, and 4 hits respectively?
We'll need to create a little procedure function to get this answer. The procedure will run through each of the games and update a separate counter every time each number of hits occurs.

Here's one possible version.

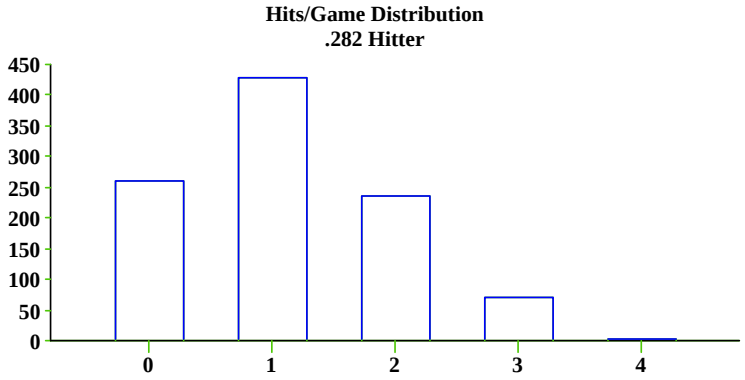
```
; The function processes a list called hits and produces the list freq.
; The 1st element of freq indicates the number of games with 0 hits,
; The 2nd element reports the number of 1-hit games... etc.

; Initiate the counter settings to 0
for j = 0 to 4
  'freq[j+1] = 0       ; no 0th element allowed in TK
next j
for i = 1 to 1000
  x = 'hits[i] + 1              ;  again, no 0th element allowed in TK
  'freq[x] = 'freq[x] + 1    ;  increments the counter
next i
```

- After you create the procedure, use the Examine Command to **call freq()**

The resulting list can be plotted as a bar chart for a visual representation of the distribution of hits per game.

hits	freq
0	261
1	429
2	237
3	71
4	2



Tools for Processing Functions

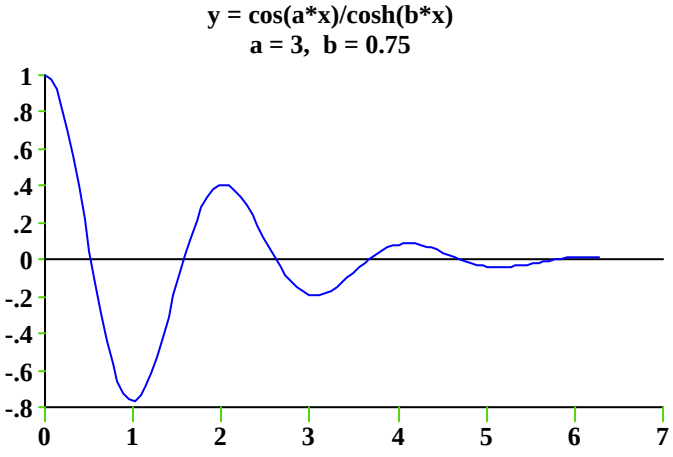
Just as we can find the square root of a variable or collate a list, we can also use TK to process functions. The TK Library has many tools for these tasks. The most commonly used tools are in the areas of differentiation, integration, optimization, and root finding. For example, we can differentiate 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.

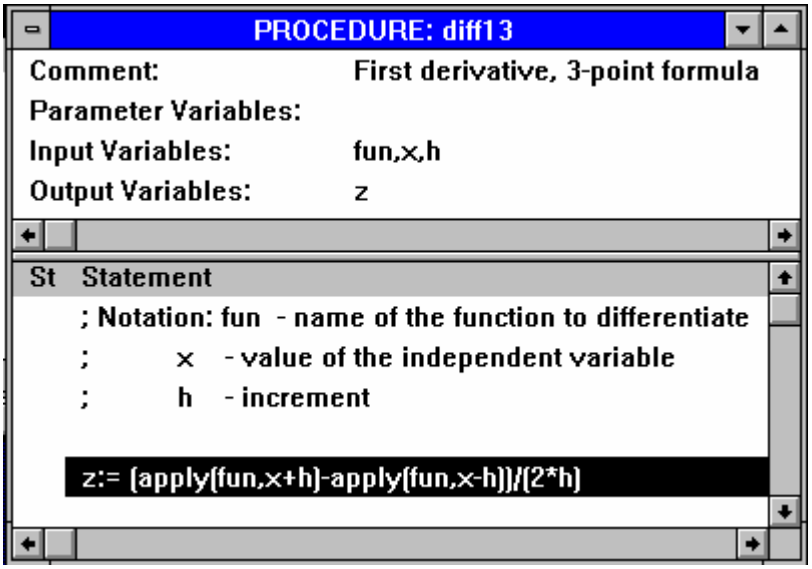
For example, suppose we have an equation such as **y = cos(a*x)/cosh(b*x)** and we are interested in studying the relationship between x and y as a and b remain constant.

- 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.

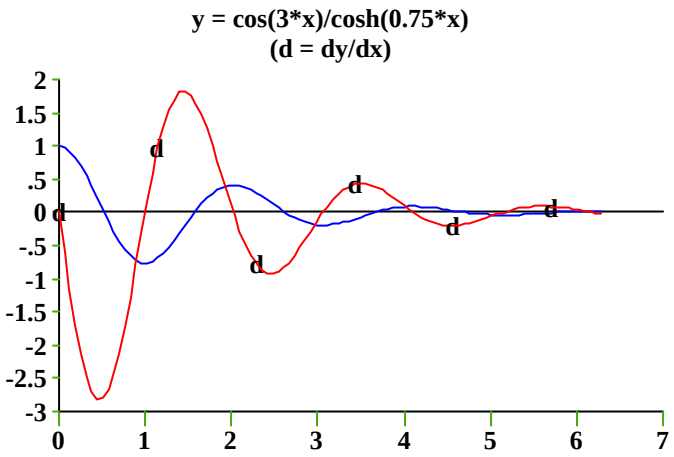
- Enter the reference to diff13 on the Rule Sheet and solve.

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.

CASE STUDY: Integration and Differentiation

A manufacturer has the capability of transforming flat sheets of metal into uniformly corrugated sheets of various pre-determined thicknesses.

The corrugation machine transforms flat sheets into rippled sheets of the form

y = 0.5*t * sin(pi()*x)

where y is the displacement of the surface of the sheet from the original horizontal plane x inches from the initial edge of the corrugation. The parameter t is the effective thickness of corrugated sheet. (To simplify this example, we assume that the material thickness is negligible.)

The plant is only able to cut flat sheets. Corrugated sheets cannot be cut, so the sheets must be shortened prior to corrugation.

Problem 1: Suppose the manufacturer currently has flat sheets in stock which are 60 inches long. What is the resulting length of the corrugated sheets if the thickness is set to 1 inch?

Use the fact that the length of a curve is defined by the equation

$$\text{length} = \int [1 + (dy/dx^2)]^{0.5}$$

where length is the length of the flat sheet and dy/dx is the derivative of y with respect to horizontal location. If the integral is done from 0 to the length of the corrugated sheet, the result will be the length of the original flat sheet.

Step 1: Define the corrugation function.

We need to compute the derivative of the corrugation function, so we first must create the function. Go to the Function Sheet and create a Rule Function called **corrugate**. Open the subsheet and enter the equation along with the variable definitions.

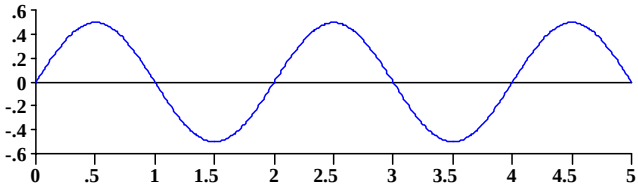
Comment: form of the corrugated sheet
Parameter Variables: **t**
Argument Variables: **x**
Result Variables: **y**

y = (t/2)*sin(pi()*x)

Close the function subsheet and go to the Rule Sheet to test it. Enter the rule

y = corrugate(x)

Switch to the Variable Sheet and enter inputs of **1** and **.2** for t and x, then solve. y = .29389263 . This means that .2 inches from the end of the corrugated sheet, the sheet is displaced almost .3 inches from the horizontal plane. Here is a TK plot of the function from x = 0 to x = 5.



Note that the curve repeats itself every 2 inches and this is independent of the thickness.

Step 2: Define the integrand of the length equation.

$$[1 + (dy/dx^2)]^{0.5}$$

This expression can be entered as a rule function. First, we need to merge in a tool to do the numeric differentiation of the corrugate function. Open the TK Library Menu and select the **Mathematics** section. From there, select the **DIFFINT** section, covering differentiation and integration. Next, select the **TOOLS** folder and merge **diff13**, 3-point numeric differentiation. The function will appear on your function sheet.

Open the subsheet for diff13.

```
Comment: First derivative, 3-point formula
Parameter Variables:
Input Variables: fun,x,h
Output Variables: z

; Notation: fun  - name of the function to differentiate
;          x    - value of the independent variable
;          h    - increment

z = (apply(fun,x+h)-apply(fun,x-h))/(2*h)
```

To compute a derivative, we need to supply diff13 the name of the function, the location, and a relatively small increment over which to do the computation. In our case, h = .001 would be appropriate.

Close the subsheet for diff13 and create a rule function called **integrand**. Open the subsheet and enter the equation and variable definitions.

```
Comment: function to be integrated
Parameter Variables:
Argument Variables: c
Result Variables: f

f = sqrt(1 + (diff13('corrugate,c,.001))^2)
```

The variables c and f represent the lengths of the corrugated and flat sheets. Close the subsheet.

Step 3: Merge and use a numerical integration tool.

Return to the TK Library Menu and merge the Simpson integration tool into your model. The function will appear on the Function Sheet. Open the function subsheet.

```
Definite integral, Simpson rule
Parameter Variables:
Input Variables: fun,a,b,n
Output Variables: value
```

Simpson requires 4 inputs -- the name of the integrand function, the integration limits, and the number of integration slices. Close the subsheet and switch to the Rule Sheet. We might use the rule

```
LF = Simpson('integrand,0,LC,100)
```

where LF and LC are the lengths of the flat and corrugated sheets. This rule uses 100 slices to do the computation. Enter the following rule to compute the integral.

```
LF = Simpson('integrand,0,LC,100)
```

Switch to the Variable Sheet, enter **60** as input for **LF** and guess **LC** as **30**. Solve, and **LC = 41** . Do you know why a guess was necessary for LC? Without it, the Simpson procedure would not have values for all of its inputs.

Now that the model is working, experiment with different input values of t and LC.

Problem 2: Consider the situation where both LF and LC are known but the thickness, t, is unknown. Let's try, for example, LC = 36 and LF = 50.

If we guess t, enter the values for LC and LF and solve, TK responds with the error message,

"Unevaluated Expression in Procedure"

Moving to the Rule Sheet, we see that the rule referencing the Simpson procedure is marked with an error symbol, >. Place the highlight over the rule and open the subsheet. We can follow the trail of error symbols through the functions until we find that the error is triggered when the diff13 function references the corrugate function with an unknown value for the parameter variable, t. This leads to several questions:

Why doesn't TK use the guess value we supplied for t?

TK initially ignores any guesses and tries to solve the problem without any guesses or iteration. It's during this initial run through the Rule Sheet that TK enter the Simpson procedure and bombs when trying to reference the diff13 procedure.

Why was t made a parameter variable in the first place?
We were anticipating the use of the diff13 function, which assumes that the function to be differentiated has one argument and one result. We differentiate a function with respect to a variable. Open the subsheet for the diff13 function and observe the use of the APPLY function. In this case, we're applying fun at x+h and x-h, the only input.

```
z = (apply(fun,x+h)-apply(fun,x-h))/(2*h)
```

Any other variables must be passed into the function as parameters. The model was originally designed to compute dy/dx, so t was passed in as a parameter.

How do we get around this predicament?
Fortunately, there is a way around the problem. We must delay the reference to the Simpson function until the guess for t has been made. To do this, use the built-in function KNOWN.

```
If known('t) then LF = Simpson('integrand,0,LC,100)
```

Try your guess again for t. Success! t = 0.89....

Important Note on Built-in Functions
Note that there are built-in functions are available for computing derivatives and integrals. For example the built-in functions DERIV and INTEGRAL can be used in the examples above. Library functions were used in the last two sections to illustrate the concept of functions processing other functions.

Root Finding -- Without TK's Iterative Solver

TK application developers may need to automate more difficult calculations to make their programs easier to use. One way of doing this is to help the user avoid the need to choose a guess for iterative calculations and then use a root-finding tool from the TK Library to do the iteration for them. The following example shows how friction factor calculations can be automated in this way.

The following Rule Sheet is set up to solve problems involving isothermal fluid flow through straight pipes. The friction factor equation must be solved iteratively because f is hopelessly embedded in both sides of the equation.

St	Rule
	1/sqrt(f) = -4*log(î/(3.7*D)+(1.255/(NRe*sqrt(f))))
	q = w/Ö
	V = 4*q/(pi()*D^2)
	NRe = V*D*Ö/æ
	Ûp/L = f*Ö*V^2/(D*16.085)
	gradient = Ûp/L
	if NRe<2000 then flowtype = 'L else flowtype = 'T

Here is the Variable Sheet with a sample set of inputs and the resulting outputs, using a guess of .001 for f. (The calculation and display units are the same for all variables.)

St	Input	Name	Output	Unit	Comment
	0.00087	æ		lb/(ft*s)	Fluid viscosity
	62.5345	Ö		lb/ft^3	Fluid density
	0.0005	î		ft	Pipe Roughness
	0.125	D		ft	Inside diameter
	21.67	w		lb/s	Weight rate of flow
	20	L		ft	Pipe length
		q	0.34653	ft^3/s	Volumetric rate of flow
		V	28.2377	ft/s	Mean linear velocity
		f	0.00721		Friction factor
		Ûp	3578.26154	lbf/ft^2	Pressure drop due to friction
		gradient	178.91308	lbf/ft^3	Pressure drop gradient
		NRe	253711.273		Reynolds number
		flowtype	T		Turbulent (T) or laminar (L) (output only)

The need for guessing and iteration when equation solving is common and should cause few problems for experienced TK users. Novices may get a bit confused as to which variables to guess and what values to start with. If we intend to share this model with others, it might be a good idea to "ruggedize" the model where possible.

One way to do this is to supply default guesses for variables which might be frequent backsolving candidates. This is done on the variable subsheet for those variables. In this case, we might supply 0.001 as a default guess for f.

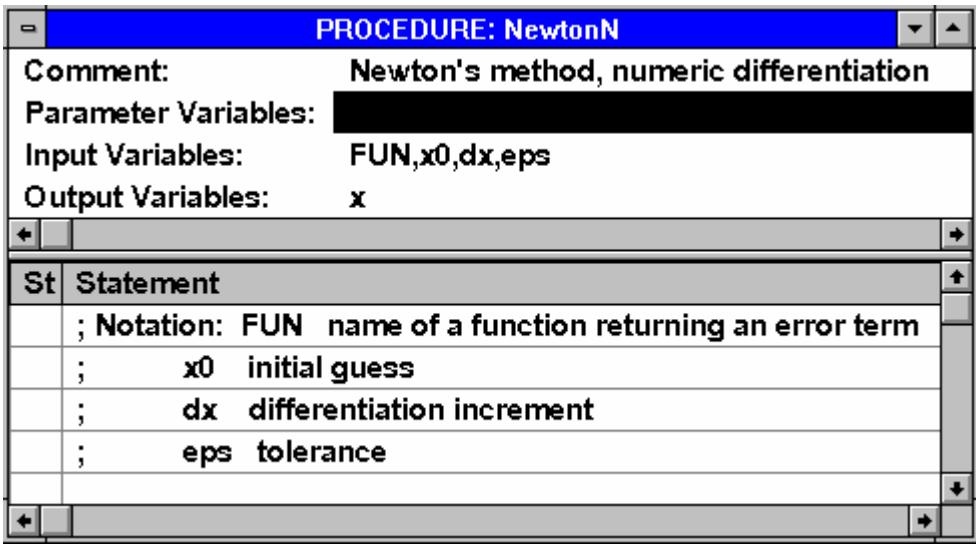
Another method is to supply dummy variables and equations to start the guess as a function of other variables. This technique was featured earlier in this booklet. One of

the drawbacks to that method is that if the model is to be backsolved for different combinations of variables, the dummy guesses will still be triggered and could cause confusion.

There is another, more rigorous method which can be used. Avoid the iterative solver altogether by using root finding tools from the TK Library.

- Open the TK Library and merge in the **newton-n** tool from the ROOTS section under Mathematics.

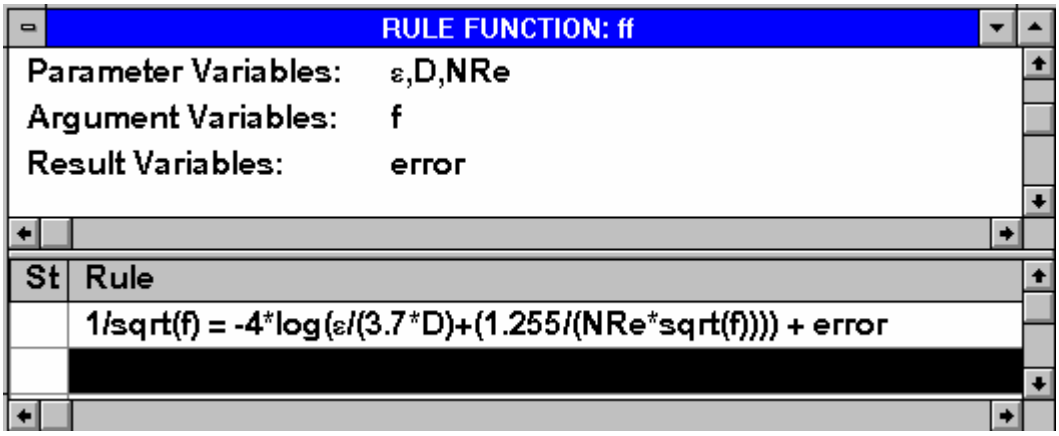
A procedure function called NewtonN will appear on the Function Sheet. NewtonN finds the root of a function using Newton's method with numeric differentiation.



NewtonN requires four inputs and returns a root of the function. The inputs are described in the function comments. Our next step is to define the function to be processed.

- Create a rule function called ff and open the subsheet.
- Copy and paste the friction factor equation from the Rule Sheet into the ff function.
- Edit the function to add an error term -- call it **error**.
- Define the Argument Variable to be **f** and the Result Variable to be **error**. All other variables in the function should be made Parameter Variables.

Your ff function should appear as follows:



- Use the Examine Command to test the function. Enter **ff(.001)** . TK returns the error value of 19.99, given .001 as the f value.

Now we can use the NewtonN procedure to find the value of **f** which makes the error value equal 0.

- Enter the following rule on the Rule Sheet.

f = NewtonN('ff,1/(-4*log((.27*ĩ/D)+(7/NRe)^.9))^2,1E-4,1E-6)

The commas separate the four expressions which are inputs to the the function. The first vaue, 'ff, points to the function to be processed. The second expression is the initial guess for f. In this case, Churchill's equation is often used to approximate the Colebrook friction factor equation, so we use it as our guess. The third value is the numeric differentiation increment, which should be relatively small. The fourth expression is the comparison tolerance which, at 1E-6, is the same as the default for TK's iterative solver.

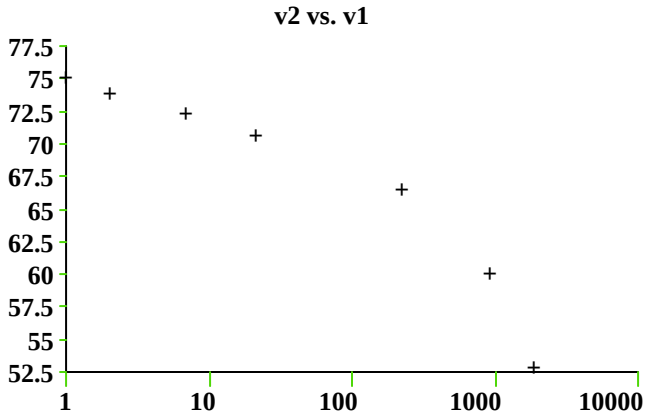
Now, when we solve, no guess is necessary on the Variable Sheet. TK uses the NewtonN function to solve directly for **f**.

Curve-Fitting

The FITSTAT section in the Statistics area of the TK Library features a variety of methods for fitting data to equations. Each of them loads with sample data and instructions. Let's try a few of them using our own data.

Case 1:

- Open the List Sheet and create two lists, v1 and v2.
- Enter the following values into the v1 and v2 lists.
  v1: 1, 2, 7, 21, 226, 917, 1864
  v2: 75.2, 73.9, 72.4, 70.7, 66.6, 60.1, 53.
- Create a line chart of v2 vs. v1. Use the symbols style with the "+" character. Change the scale to X-log. View the plot.



Hmmm... based on the plot, it looks like we might be able to fit a quadratic or cubic equation through the points, if we first apply a log transformation to the v1 values. We can use the Polynomial Regression routine in the TK Library to do the job.

- Our first step is to store our data to an ASCII file, using the File Export Command. Call the file data1.asc. We do this for several reasons: (1) we can access the data later from other programs if necessary; (2) we avoid the chance of overwriting or losing the data when we load a curve-fitting routine.

- Next, open the Library Menu and load the Polynomial Regression routine from the FITSTAT section under Statistics. Choose the Reset and Load option.

The Polynomial Regression routine displays a "works" table of inputs and outputs. The data in the table is a sample set. The column on the right is where we set the order of the polynomial to fit to the data. The rest of the values in that column are outputs. The solution equation is b1 + b2*X + b3*X^2 + ...

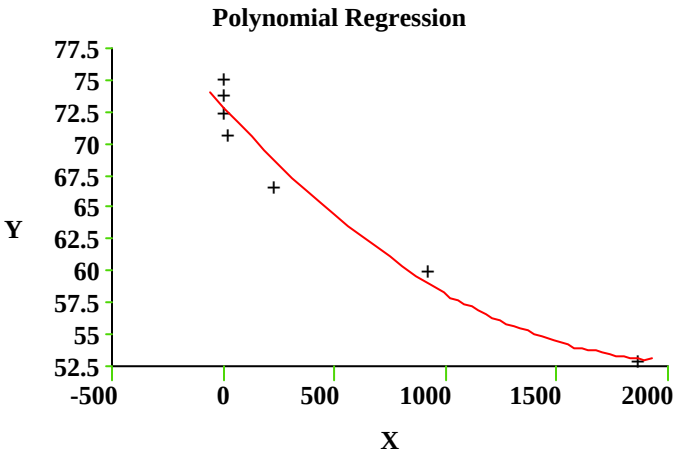
Element	X	Y	residuals	SUMMARY	STATS
1	1.2	42		order	2
2	1.5	34		N	
3	1.8	30		Syx	
4	2.1	29		adj R2	
5	2.4	28		p	
6	2.7	24			
7	3	15		b1	
8	3.3	6		b2	
9	3.6	.8		b3	
10	3.9	.5		b4	

We need to import our data and get it into the X and Y lists in the table.

- Use File Import to load data1.asc. The lists will appear at the bottom of the List Sheet. Go to the List Sheet and blank the names of the lists X and Y. Then change the names of the v1 and v2 lists to X and Y (be sure to use capital letters!). Close the List Sheet and open the "works" table again.

Element	X	Y	residuals	SUMMARY	STATS
1	1	75.2		order	2
2	2	73.9		N	
3	7	72.4		Syx	
4	21	70.7		adj R2	
5	226	66.6		p	
6	917	60.1			
7	1864	53		b1	

- Keep the order set at 2 and solve. The adjusted R^2 statistic reports the goodness of fit as .94. Check the resulting plot.



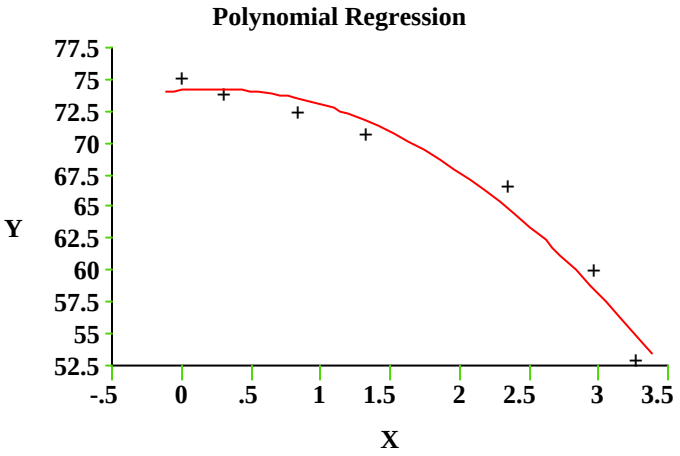
It's ok but we might be able to do better if take the log of the X values before solving. In this case, with only 7 values, we could easily edit each of the values manually. But what if we had 50 values or 500 values? A procedure function would be easier. Let's try that just for practice.

- Create a procedure function called logger and enter the following statements.

Statement
for i = 1 to 7
'X[i] = log('X[i])
next i

- Call the function using the Examine Command, **call logger().** The X column of the table is now updated.

Solve again and observe the plot. The adjusted R^2 value increased slightly to .95.



This plot is more informative.

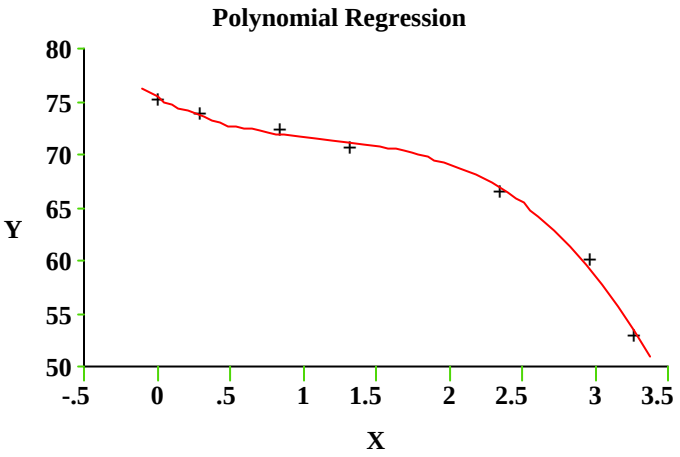
- Let's try a third order polynomial and see if it improves the fit significantly.

Element	X	Y	residuals	SUMMARY	STATS
1	0	75.2	-2.363E-1	order	3
2	.30102999566	73.9	+2.363E-1	N	7
3	.84509804001	72.4	+4.084E-1	Syx	.648517182
4	1.3222192947	70.7	-4.884E-1	adj R2	.993695615
5	2.3541084391	66.6	-2.066E-1	p	.002270867
6	2.9623693357	60.1	+7.182E-1		
7	3.270445908	53	-4.316E-1	b1	75.4362675
8				b2	-7.2702976
9				b3	5.03902749
10				b4	-1.4901058

The adjusted R2 value increased to .99 and the plot shows that the fit is much better. The final equation, with respect to our original data, is

v2 = b1 + b2*log(v1) + b3*(log(v1))^2 + b4*(log(v1)^3)

Note that we should be careful in using the resulting equation to extrapolate beyond the range of the current data. Higher order polynomials can change rapidly.



Afterthoughts:

In many cases, the Polynomial Regression routine makes the best first choice for curve-fitting. You can quickly enter your data in the table and get results. In this example, we took a slower path to expose you to the important issue of data handling. There was no need to enter, plot, and store the data separately.

What if the log transformation didn't help with the fit? We could make a quick change to the logger function to transform the data back to their original values.

'X[i] = 10^('X[i])

The Examine Command could be invoked again to call the function.

Nonlinear Curve-Fitting

In many cases, the form of the equation used to fit the data is already known from past experience. The task is to determine the coefficients which make the equation fit as well as possible. The following example shows how this is done in TK.

- Use the Menu to access the Statistics section of the Library. From the FITSTAT area, load the routine labeled Nonlinear Curve-Fitting, One Predictor Variable. The instructions are presented in the form of a Comment Sheet.

The equation we need to fit to our data is

$$y = D + \frac{A}{1 + e^{(B \cdot (x - C))}}$$

- According to the instructions, we need to enter this as the definition in the procedure function called FUNCTION. Following the prescribed syntax conventions for the function, we enter it as shown below.

St	Statement
	for i=1 to n
	y[i] = b[4] + b[1]/(1+exp(b[2]*(x[i]-b[3])))
	next i

All of the variables from our original equation have been replaced by references to list elements which are processed by the curve-fitting routine. A is now b[1], B is b[2], C is b[3] and D is b[4]. Each data point is represented by the ith element of a list, y[i] and x[i]. Apostrophes are not used in this function, indicating that the function can be used to process different lists in the course of reaching a solution and plotting the results. The names of the lists are passed into the function as input variables.

- The next step is to enter the data in the table called spec. We also enter initial guesses and bounds for the unknown coefficients in the b0, bmin and bmax columns. The b and SSE columns are reserved for output.

If you already have data stored in a file or another application, you must get it into TK lists called x and y (lower-case) in order for it to be processed by this routine. You can copy and paste it directly from a spreadsheet into the table. If you are importing the data, go to the List Sheet and change or blank the names of the existing x and y lists and then rename your lists.

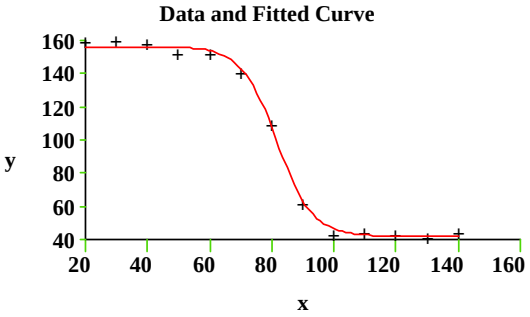
Element	x	y	b0	bmin	bmax	b	SSE
1	20	158.81	100	0	1000		
2	30	159.5	.1	-1	1		
3	40	157.91	100	0	1000		
4	50	151.38	100	-1000	1000		
5	60	151.88					
6	70	140.69					
7	80	109.74					
8	90	62.12					
9	100	42.95					
10	110	44.44					
11	120	43.2					
12	130	41					
13	140	43.73					

- Solve and watch the status bar at the bottom of the TK Window for a progress report in the form of the sum of squared errors between the actual and predicted y values as TK tries new b coefficients.

After several steps, the results are added to the table. The SSE value decreased from over 9000 to 76.6 during the process.

Element	x	y	b0	bmin	bmax	b	SSE
1	20	158.81	100	0	1000	114.2769	76.60439
2	30	159.5	.1	-1	1	.1762495	
3	40	157.91	100	0	1000	81.63323	
4	50	151.38	100	-1000	1000	41.99055	
5	60	151.88					
6	70	140.69					
7	80	109.74					
8	90	62.12					
9	100	42.95					
10	110	44.44					
11	120	43.2					
12	130	41					
13	140	43.73					

- Now let's see how well our equation fits the data by viewing the plot called fit.



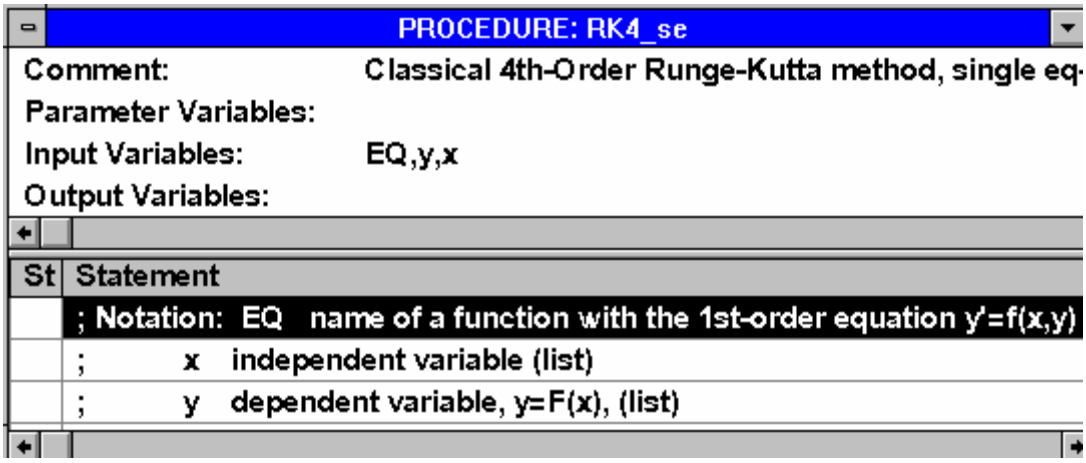
First Order Nonlinear Ordinary Differential Equations

Suppose we need to solve the following first order nonlinear ordinary differential equation for y, given y(0) = -2.0 and 0 <= x <= 1

$$y` = e^{(x)} \cdot y^2 - y + e^{(x)}$$

We can solve the problem numerically using the RK4_se function from the TK Library. The "se" extension stands for single equation. There are other library functions adapted for multiple-equation problems.

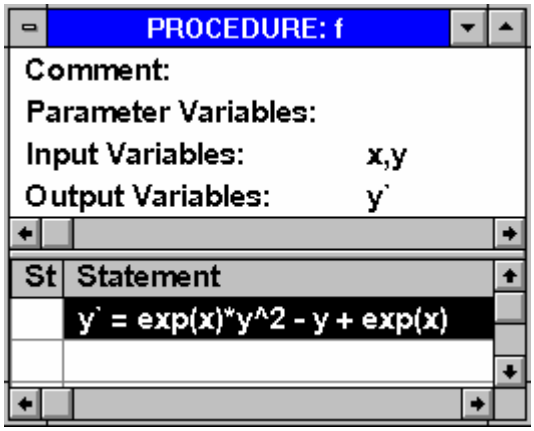
- Use the Library Menu to load the tool labeled rk4se from the DIFFEQ section under Mathematics. The RK4_se function loads on the Function Sheet.



As indicated by the comments, the procedure requires three inputs in the form of a function name and two list names.

Create a new procedure function called f and enter the following information.

It is important to note that the "`" in the expression y` is not an apostrophe but a prime symbol. There really is no need to use a prime symbol but it is frequently used in refering to a derivative of a function.



The next step in solving the problem is to create a table for the inputs and results. Specifically, we need to provide a list of values for the independent variable, x, along with the initial condition and room for the dependent variable, y.

- Make a table with two lists, x and y, and the information shown, using the List Fill Command to save data entry time. For x, fill the list from 0 to 1 using step sizes of 1/32. A portion of the table is shown below.

Element	x	y
1	0	-2
2	.03125	
3	.0625	
4	.09375	
5	.125	

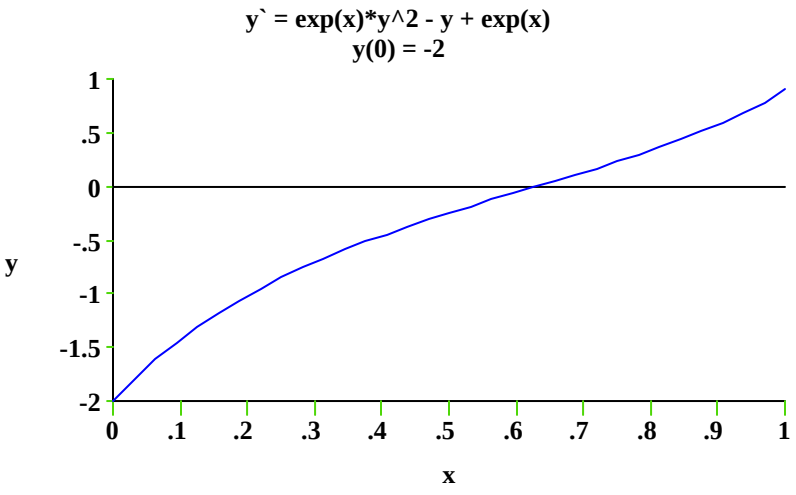
The next step is create a rule to call the rk4_se function with the proper inputs values.

call RK4_se('f,'y,'x)

When TK processes this rule, it fills the y values in the table. Here is a portion.

Element	x	y
28	.84375	.432240249
29	.875	.507691307
30	.90625	.58964571
31	.9375	.680059255
32	.96875	.781583146
33	1	.897931729

The solution can also be displayed as a line chart.



Now let's backsolve this problem, specifying that y(1) = 1, and solve for the initial condition, y(0), which causes such a solution. That is, when x = 1, y = 1. Our job is to find y when x = 0. Currently, y(1) = 0.8979.

- Edit the Rule Sheet as shown below.

Rule
place('y,1) = yg
if known('yg) then call RK4_se('f,'y,'x)
if known('yg) then 'y[33] = 1

The first rule now place the value of the variable yg into the first element of the list y. yg will be a guess variable on the Variable Sheet. TK will iterate on the value of yg until all the rule are consistent.

The second rule is the same as before except that the known function prevents TK from calling the procedure until the guess is known.

The third rule again checks to see if the guess is known and, if so, equates the 33rd element (where x = 1) of the y list to 1.

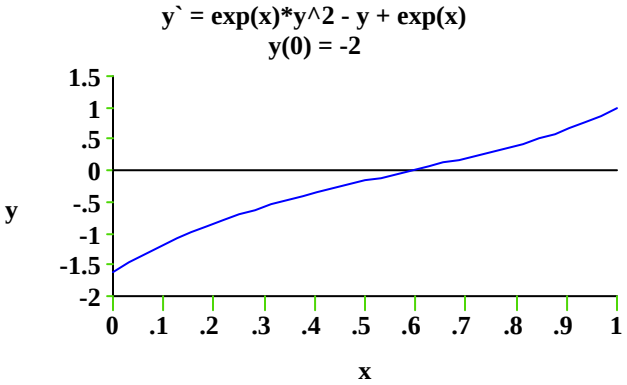
The use of the known function is vital when backsolving procedures which call other functions. It prevents the procedure from getting into error conditions.

Open the Variable Sheet, enter an initial guess of -2, and solve. After several iterations, TK converges to the solution, y(0) = -1.6276...

Element	x	y
1	0	-1.6276275

...

Element	x	y
33	1	1



Bilinear Interpolation

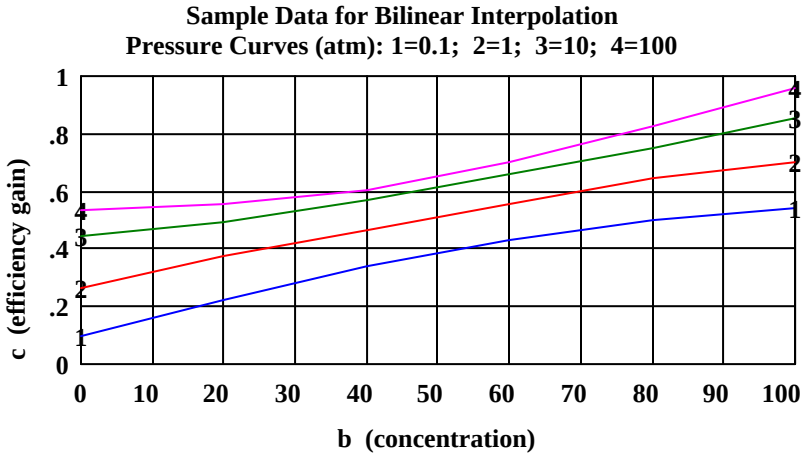
Many TK Solver applications involve interpolations within families of curves. Handbooks and design guides frequently provide such curves. Unfortunately, without equations describing the relationships shown on these plots, we are left to extract the data numerically. This example illustrates how to set up and solve these problems using bilinear interpolation.

A fictitious group of scientists invented a new fuel injection system which improves the performance of combustion engines. Additionally, the group discovered that further enhancement can be achieved by using their secret "air additive", by itself or mixed with the air in the chamber. The effectiveness of the new system depends on both the concentration of the additive in the air and the pressure of the air/additive mixture.

The scientists reported the performance of the system including a matrix and a plot describing the relationship between the pressure, concentration, and performance enhancement. The variables are defined as follows:

a = pressure, atm; values are 0.1, 1, 10, and 100
b = gas concentration, %; values range from 0 to 100
c = performance enhancement

Element	b	c @ a1	c @ a2	c @ a3	c @ a4
1	0	.095	.262	.441	.532
2	20	.221	.377	.491	.553
3	40	.338	.467	.569	.607
4	60	.433	.559	.661	.7
5	80	.501	.647	.748	.829
6	100	.545	.701	.856	.96



Our goal will be to computerize this chart so that for any given pressure or concentration within the domain of values, the value of c is evaluated.

- Step 1: Create a table like the one shown and enter the values. Here is the Table Subsheet required.

List	Format	Width	Heading
b		6	b
c1		6	c @ a1
c2		6	c @ a2
c3		6	c @ a3
c4		6	c @ a4

- Step 2: Merge in the inpol2 function from the TK Library. It's in the Utilities Section under SOLVUTIL TOOLS. When it's loaded open the subsheet for the description and information on the necessary inputs.

St	Statement
	; Notation: x,y - arguments
	; z - interpolated function value
	; xname - master list of lists with z=f(y) values
	; xlist - list of x values
	; ylist - list of y values
	; Description: This procedure searches for a cell that includes given x and
	; y , and performs bilinear interpolation. If the given x,y fall on a grid
	; line or grid point, the procedure performs simple linear interpolation or
	; copies the table value. It handles incomplete tables with missing values
	; in corner regions. It indicates an error when the given x,y are outside
	; the definition interval.

We need to supply the function with 5 inputs and it returns 1 result, the interpolated value. The first two variables, x and y, will be mapped to given values of pressure (a) and concentration (b) in our example. The variable xname refers to a list of list names -- in our case, the four c columns. We must create a list to hold the column names.

- Step 3: Go to the List Sheet, create a list called c, and fill it with the four list names c1 through c4. Don't forget the apostrophes.

List: c
'c1
'c2
'c3
'c4

Next the function asks for the name of the list holding the x values. These are the pressure values. Currently, this list does not exist.

- Step 4: Go to the List Sheet, create a list called a and fill it with the pressure values: .1, 1, 10, and 100.

Next, the function requires the name of the list of y values. These are the concentrations in the list b, already set up in the table.

- Step 5: Enter a rule on the Rule Sheet referencing the inpol2 function with the proper inputs.

  C = inpol2(A,B,'c,'a,'b)

- Step6: Go to the Variable Sheet and enter .5 for A, 35 for B, and solve.

Sta	Input	Name	Output	Unit	Comment
					Bilinear Interpolation Example
	.5	A		atm	Pressure
	35	B		%	Concentration
		C	.36908333		Combustion Efficiency Improvement

The resulting C value is .369. Checking the plot, this value seems ok... but we can do much better.

Currently, the model is using linear interpolation between the pressure curves but the pressure values are not linearly spaced. They are on a log scale. For example, if we go half way between the first and second pressure curves, are we half way from .1 to 1, at a pressure of .55? It is more accurate to interpolate on the logs of the pressure values and then transform the results. In this case, we would be halfway between log(.1) = -1 and log(1) = 0, which is -.5. Then, converting the interpolated value back to linear scale, the pressure value is 10^-.5 = .3162278. This is significantly different from the .55 value used without the transformation. Let's make the necessary changes to the model.

- Step 1: Go to the List Sheet and edit the values in the list a. Change them to log values -- -1, 0, 1, and 2.

- Step 2: Edit the rule on the Rule Sheet to convert the input for A to its log and solve.

  C = inpol2(log(A),B,'c,'a,'b)

Sta	Input	Name	Output	Unit	Comment
					Bilinear Interpolation Example
	.5	A		atm	Pressure
	35	B		%	Concentration
		C	.40363518		Combustion Efficiency Improvement

The new solution for C is .403... The same log transformation technique can be used for all three variables involved, if necessary.

The inpol2 function can be backsolved with iteration but can be sensitive to the the choice of guess used and "bounce" out of the domain of the data.