[6.22] Linear Interpolation

Linear interpolation is the least sophisticated, least accurate interpolation method. However, it has the advantages that it is fast, and often accurate enough if the table values are closely spaced. Given a table of function values like this:

x1	y1
x	y
x2	y2

where *x1*, *y1*, *x2* and *y2* are given, the problem is to estimate *y* for some value of *x*. In general,

$$y = y1 + (y2 - y1)\left[\frac{x - x1}{x2 - x1}\right]$$

which can be derived by finding the equation for the line that passes through (x1,y1) and (x2,y2).

If you interpolate often, it is worth defining this as a function. At the command line:

```
define interp(xx1,yy1,xx2,yy2,x)=yy1+(yy2-yy1)*((x-xx1)/(xx2-xx1))
```

or in the program editor:

```
interp(xx1,yy1,xx2,yy2,x)
Func
©Linear interpolation
yy1+(yy2-yy1)*((x-xx1)/(xx2-xx1))
EndFunc
```

Note that I use variable names of the form *yy1* to avoid contention with the built-in function variables *y1* and *y2*. If xx1 = 1, yy1 = 1, xx2 = 2 and yy2 = 4, either of these routines returns y = 2.5 for x = 1.5.

Either of these methods are fast and work well. However, you have to remember the order of the input variables. One way to avoid this is to use the Numeric Solver. This program automates the process:

```
linterp()
Prgm
©Linear interpolation with numeric solver

©Delete equation variables
delvar x,y,xx1,xx2,yy1,yy2

©Save interpolation equation
y=((yy1-yy2)*x+xx1*yy2-xx2*yy1)/(xx1-xx2)→eqn

©Start the numeric solver
setMode("Split 1 App","Numeric Solver")
EndPrgm
```

This program works by saving the interpolation equation to the system variable *eqn*, which is used by the numeric solver. All the equation variables are deleted first, otherwise any current values will be substituted into the equation and it will be simplified, which won't give the correct equation to solve. Finally, the numeric solver is started with the last program line.

To use this program, enter *linterp()* at the command line, then press ENTER at the solver *eqn:* prompt. The prompts for the six variables are shown. Enter the required values for *x*, *xx1*, *xx2*, *yy1* and *yy2*, then solve for *y*.

Another advantage of this program is that it is easy to find *x*, given *y*. This process is sometimes called inverse interpolation. You can also use the *interp()* functions above for inverse interpolation: just enter the y-values as x-values, then solve for *y* as usual, which will actually be the interpolated value for *x*.