[6.31] Fast 2nd-order interpolation with *QuadReg*

Linear interpolation is adequate for most table-interpolation problems. A second-order (quadratic) interpolation is more accurate, but it does require entering one more data point. The first-order (linear) interpolation uses two points to find the coefficients *a* and *b* in

$$ax + b = y$$

then calculates *y* at your desired *x*. A second-order interpolation uses three points to find *a*, *b* and *c* in

$$ax^2 + bx + c = y$$

and again finds *y* at your desired *x*. The coefficients can be found as those of a Lagrangian interpolating polynomial, but that is not built-in to the 89/92+. Quadratic regression is built-in, and it can be used to find the coefficients. The basic steps are

1. Save the x- and y-coordinates for the three points to two list variables, one for the x-coordinates, and one for the y-coordinates.
2. Execute the QuadReg command with the two lists.
3. Use the built-in system function RegEq(x) to interpolate y = f(x)

As an example, suppose that we have a table of gamma function values for x=1 to x=2, spaced every 0.04. We want to find gamma(1.895). We choose the three table x-values which bracket our desired x-value of 1.895, where the middle x-value is the one closest to the desired x-value. From the table, the three points are

x1 = 1.84	y1 = 0.94261236	
x2 = 1.88	y2 = 0.95507085	*x2=1.88 is closest to x=1.895*
x3 = 1.92	y3 = 0.96877431	

Then these steps estimate gamma(1.895):

```
{1.84,1.88,1.92}→lx
{.94261236,.95507085,.96877431}→ly
QuadReg lx,ly
regeq(1.895)
```

which returns gamma(1.895) = 0.96006375. The actual answer is 0.960062793, for an error of about 9.6E-7. Linear interpolation results in an error of about 1.5E-4, so the 2nd-order interpolation gives several more significant digits.

Note that the *regeq*() function is a system variable that is updated when any regression command, including *QuadReg*, is executed. Once you have executed *QuadReg*, you can use *regeq()* to interpolate for additional x-values, as needed. These values need to be between *x1* and *x2*, or you are not interpolating, you are extrapolating, which is much less accurate.

This method cannot be written as a function because the *QuadReg* command only accepts global list variable names as arguments. The *QuadReg* arguments *must* be list names, and not the lists themselves.

I built this method into a program called *QuadInt(),* which is shown here:

```
quadint()
Prgm
©2nd-order interpolation
©15oct00 dburkett@infinet.com
```

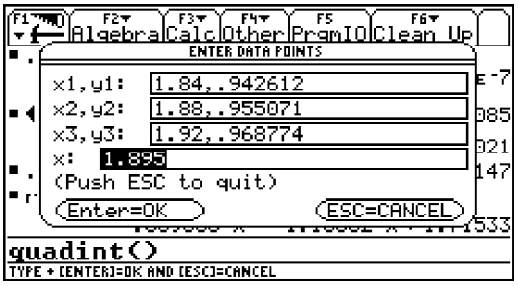
```
©calls str2var()

local a1,b1,a2,b2,a3,b3,x1y1,x2y2,x3y3,x

© Initialize all the point variables
© (note that a1==x1, b1==y1, etc)
1→a1
2→b1
3→a2
4→b2
5→a3
6→b3
Ø→x

© Transfer control here to repeat interpolations
lbl t1

© Convert x- and y-data to pairs as strings; also convert 'x' to string
string(a1)&","&string(b1)→x1y1
string(a2)&","&string(b2)→x2y2
string(a3)&","&string(b3)→x3y3
string(x)→x

© Dialog box for user to enter xy data points
dialog
 title "ENTER DATA POINTS"
 request "x1,y1",x1y1
 request "x2,y2",x2y2
 request "x3,y3",x3y3
 request "x",x
 text "(Push ESC to quit)"
enddlog
if ok=Ø:goto exit1                    © Give user a chance to quit here

© Convert the interpolation 'x' to a number
expr(x)→x

© Convert the x1, y1 data point string to numbers
str2var(x1y1)→res
if gettype(res)="STRING":goto err1
res[1]→a1
res[2]→b1

© Convert the x2, y2 data point string to numbers
str2var(x2y2)→res
if gettype(res)="STRING":goto err1
res[1]→a2
res[2]→b2

© Convert the x3, y3 data point string to numbers
str2var(x3y3)→res
if gettype(res)="STRING":goto err1
res[1]→a3
res[2]→b3

© Build lists for QuadReg call
{a1,a2,a3}→l1
{b1,b2,b3}→l2

© Do the regression and calculate y = f(x)
quadreg l1,l2
regeq(x)→res

© Display the result
dialog
```

```
 title "QUADINT RESULT"
 text "x = "&string(x)
 text "y = "&string(res)
 text "y saved in 'res'"
 text "(Push ESC to quit)"
enddlog
if ok=0:goto exit1                    © Give user a chance to quit here
goto t1

© End up here, if user forgets to separate values with commas
lbl err1
dialog
 text "x and y values must"
 text "be separated with commas"
enddlog
goto t1                               © Go try again

© Delete global variables before exiting
lbl exit1
delvar l1,l2

EndPrgm
```

This program just provides a convenient interface for entering the xy-data points and displaying the result. Since the user might want to do several interpolations before quitting, the program loops to repeat the interpolations. The user can press [ESC] at any dialog box to quit the program. The interpolation result is saved in a global variable *res*, which can be recalled at the home screen after exiting *quadint().* The global list variables are deleted when the user quits the program.

quadint() calls the function *str2var()* to process the user's input. *str2var()* must be in the same folder as *quadint(),* but they may be in any folder. You must make that folder current with *setfold()* before running *quadint(). str2var()* is shown at the end of this tip.

When the program is executed, this dialog box is shown:



Each xy-data pair is entered in a single *Request* field; see tip [9.13] for more details on this. The x- and y-data values are separated with a comma. An error message is shown if the user forgets to use the comma. The picture above shows the fields filled in with the values for the example above. When all the values are entered, the user presses [ENTER] to do the interpolation. Another dialog box is shown with the interpolation result.

You can use this method with any of the built-in regression commands. You need to enter as many points as there are coefficients in the equation. For example, the *CubicReg* command fits the data to a cubic polynomial with four coefficients, so you would need to enter four points.

I mentioned that the function *str2var()* is called by *quadint().* This is *str2var():*

```
str2var(xy)
Func
©("x,y") returns {x,y}
©15oct00 dburkett@infinet.com

local s,x,y

instring(xy,",")→s              ©Find comma
if s=0:return "str2var err"     ©Return error string if no comma found

expr(left(xy,s-1))→x            ©Get 'x' from string
expr(right(xy,dim(xy)-s))→y     ©Get 'y' from string

return {x,y}                    ©Return list of x & y

EndFunc
```