[6.28] Bilinear interpolation

Tip [6.22] shows how to do linear interpolation in two dimensions, that is, to estimate y = f(x), given two points. This basic concept can be extended to three dimensions, in that we want to estimate z = f(x,y). In this tip, I show two methods two interpolate in three dimensions. The first method, called bilinear interpolation, requires four (x,y,z) points and performs a linear interpolation. The second method, which I call 9-point interpolation, requires nine (x,y,z) points, but fits the data to a general second-order polynomial in *x* and *y.* This method is slower, but more accurate.

Bilinear interpolation

If we are given four (x,y) points such that

$f(x_1,y_1) = z_1$ and $x_2 > x_1$
$f(x_1,y_2) = z_2$ $y_2 > y_1$
$f(x_2,y_1) = z_3$
$f(x_2,y_2) = z_4$

then we can solve for a function in four unknown coefficients. There are an infinite number of such equations, but the most simple candidate is

$$z = ax + bx + cxy + d$$

so the system to solve for *a*, *b*, *c* and *d* is

$$ax_1 + by_1 + cx_1y_1 + d = z_1$$
$$ax_1 + by_2 + cx_1y_2 + d = z_2$$
$$ax_2 + by_1 + cx_2y_1 + d = z_3$$
$$ax_2 + by_2 + cx_2y_2 + d = z_4$$

We could solve this directly for *a*, *b*, *c* and *d*, but we can get a more simple solution by scaling x and y, like this:

$$t = \frac{x-x_1}{x_2-x_1} \quad \text{and} \quad u = \frac{y-y_1}{y_2-y_1}$$

With this scaling, t = 0 when $x = x_1$ and t = 1 when $x = x_2$. Similarly, u = 0 when $y = y_1$ and u = 1 when $y = y_2$. This equation system to solve simplifies to this:

$d = z_1$ or $a = z_3 - z_1$
$b + d = z_2$ $b = z_2 - z_1$
$a + d = z_3$ $c = z_1 - z_2 - z_3 + z_4$
$a + b + c + d = z_4$ $d = z_1$

So the equation to estimate *z* in terms of *t* and *u* is

$$z = (z_3 - z_1)t + (z_2 - z_1)u + (z_1 - z_2 - z_3 + z_4)ut + z_1$$

We can expand this equation, collect on z_1, z_2, z_3 and z_4, and further factor that result to get

$$z = z_1(1-u)(1-t) + z_2u(1-t) + z_3t(1-u) + z_4ut$$

The function *bilinint()* does the interpolation with this formula

```
bilinint(xa,xb,ya,yb,zaa,zab,zba,zbb,x,y)
```

```
Func
©(xa,xb,ya,yb,zaa,zab,zba,zbb,x,y) Lin interpolate z=f(x,y)
©8oct00 dburkett@infinet.com

local t,u

(x-xa)/(xb-xa)→t
(y-ya)/(yb-ya)→u

(1-t)*(1-u)*zaa+t*(1-u)*zba+t*u*zbb+(1-t)*u*zab

EndFunc
```

Note that the variable names have been changed to avoid conflict with built-in variables. The equivalence is

xa == x_1	ya == y_1	zaa == z_1	zab == z_2
xb == x_2	yb == y_2	zba == z_3	zbb == z_4

The z-variables are named so it is easy to remember which z-variable goes with a particular (x,y) data point. The second two characters of each z-variable indicate the corresponding x- and y-variables:

zaa = f(xa,ya) zab = f(xa,yb) zba = f(xb,ya) zbb = f(xb,yb)

As an example, given this data:

	ya = 0.2	yb = 0.3
xa = 0.5	zaa = 0.4699	zab = 0.4580
xb = 0.6	zba = 0.5534	zbb = 0.5394

to interpolate for x = 0.52 and y = 0.28, the call is

```
bilinint(.5,.6,.2,.3,.4699,.4580,.5534,.5394,.58,.28)
```

which returns 0.4767.

This program, *bilinui(),* provides a user interface for the interpolation function.

```
bilinui()
Prgm
©User interface for bilinint()
©8oct00 dburkett@infinet.com

local z

if gettype(x1)="NONE" then
 0→xa:2→xb:0→ya:2→yb
 0→zaa:1→zab:2→zba:3→zbb
 1→x:1→y
endif

lbl l1

string(xa)→xa:string(xb)→xb
string(ya)→ya:string(yb)→yb
string(zaa)→zaa:string(zab)→zab
string(zba)→zba:string(zbb)→zbb
```

```
dialog
 title "bilinui"
 request "x1",xa
 request "x2",xb
 request "y1",ya
 request "y2",yb
 request "f(x1,y1)",zaa
 request "f(x1,y2)",zab
 request "f(x2,y1)",zba
 request "f(x2,y2)",zbb
enddlog

expr(xa)→xa
expr(xb)→xb
expr(ya)→ya
expr(yb)→yb
expr(zaa)→zaa
expr(zab)→zab
expr(zba)→zba
expr(zbb)→zbb

if ok=0:return

string(x)→x
string(y)→y

dialog
 title "bilinui"
 request "x",x
 request "y",y
enddlog

expr(x)→x
expr(y)→y

if ok=0:return

bilinint(xa,xb,ya,yb,zaa,zab,zba,zbb,x,y)→z

dialog
 title "bilinui result"
 text "z= "&string(z)
enddlog

if ok=0:return

goto l1

EndPrgm
```

This program accepts the user input in dialog boxes, and displays the result in a dialog box. The previous inputs are saved in global variables, so they can be used as defaults the next time you run the program. These global variables are

xa, xb, ya, yb, zaa, zab, zba, zbb, x, y

You can press [ESC] at any dialog box to exit the program. Otherwise, the program continues to run, so that you can do more than one interpolation at once.

Note that the input must be done in two dialog boxes, since there are ten input elements, and a single dialog box can hold only eight. The program calls *bilinint()* to perform the actual interpolation. *bilinui()* and *bilinint()* must be in the same folder, and that folder must be the current folder.

This method is good enough for many problems, assuming that the function changes slowly, and the x- and y-values are close enough together such that the function is 'close' to linear in both variables. The interpolated values change smoothly and gradually within a particular x and y pair, but the derivatives change abruptly as you move to the next row or column of the original data table. This may or may not matter in your application. If it matters, there are more sophisticated interpolation methods which force a continuous derivative across row and column boundaries. For more discussion, refer to

http://lib-www.lanl.gov/numerical/bookfpdf/f3-6.pdf

which is the link to the appropriate section in the book *Numerical Recipes in Fortran*.

9-point interpolation

Bilinear interpolation cannot account for any curvature in the function to be interpolated, since it is, by definition, linear. The 9-point interpolation method essentially performs a Lagrangian interpolating polynomial fit on the function

$$z = a \cdot x^2 \cdot y^2 + b \cdot x^2 \cdot y + c \cdot x \cdot y^2 + d \cdot x^2 + e \cdot y2 + f \cdot x \cdot y + g \cdot x + h \cdot y + i \qquad [1]$$

Since the interpolating polynomial includes squared terms of each independent variable, it can match some curvature in the underlying function.

We need to find the coefficients *a, b, c ...*, then we can calculate *z* for *x* and *y.* We have nine equations in nine unknowns if we use the points in the table function to interpolate like this

	y_1	y_2	y_3
x_1	z_1	z_2	z_3
x_2	z_4	z_5	z_6
x_3	z_7	z_8	z_9

We choose x_2 and y_2 as the closest values in the table to the *x* and *y* at which we want to interpolate. This results in a linear system of nine equations:

$$z_1 = a \cdot x_1^2 \cdot y_1^2 + b \cdot x_1^2 \cdot y_1 + c \cdot x_1 \cdot y_1^2 + d \cdot x_1^2 + e \cdot y_1^2 + f \cdot x_1 \cdot y_1 + g \cdot x_1 + h \cdot y_1 + j \qquad [2]$$

$$z_2 = a \cdot x_1^2 \cdot y_2^2 + b \cdot x_1^2 \cdot y_2 + c \cdot x_1 \cdot y_2^2 + d \cdot x_1^2 + e \cdot y_2^2 + f \cdot x_1 \cdot y_2 + g \cdot x_1 + h \cdot y_2 + j$$

$$z_3 = a \cdot x_1^2 \cdot y_3^2 + b \cdot x_1^2 \cdot y_3 + c \cdot x_1 \cdot y_3^2 + d \cdot x_1^2 + e \cdot y_3^2 + f \cdot x_1 \cdot y_3 + g \cdot x_1 + h \cdot y_3 + j$$

$$z_4 = a \cdot x_2^2 \cdot y_1^2 + b \cdot x_2^2 \cdot y_1 + c \cdot x_2 \cdot y_1^2 + d \cdot x_2^2 + e \cdot y_1^2 + f \cdot x_2 \cdot y_1 + g \cdot x_2 + h \cdot y_1 + j$$

$$z_5 = a \cdot x_2^2 \cdot y_2^2 + b \cdot x_2^2 \cdot y_2 + c \cdot x_2 \cdot y_2^2 + d \cdot x_2^2 + e \cdot y_2^2 + f \cdot x_2 \cdot y_2 + g \cdot x_2 + h \cdot y_2 + j$$

$$z_6 = a \cdot x_2^2 \cdot y_3^2 + b \cdot x_2^2 \cdot y_3 + c \cdot x_2 \cdot y_3^2 + d \cdot x_2^2 + e \cdot y_3^2 + f \cdot x_2 \cdot y_3 + g \cdot x_2 + h \cdot y_3 + j$$

$$z_7 = a \cdot x_3^2 \cdot y_1^2 + b \cdot x_3^2 \cdot y_1 + c \cdot x_3 \cdot y_1^2 + d \cdot x_3^2 + e \cdot y_1^2 + f \cdot x_3 \cdot y_1 + g \cdot x_3 + h \cdot y_1 + j$$

$$z_8 = a \cdot x_3^2 \cdot y_2^2 + b \cdot x_3^2 \cdot y_2 + c \cdot x_3 \cdot y_2^2 + d \cdot x_3^2 + e \cdot y_2^2 + f \cdot x_3 \cdot y_2 + g \cdot x_3 + h \cdot y_2 + j$$

$$z_9 = a \cdot x_3^2 \cdot y_3^2 + b \cdot x_3^2 \cdot y_3 + c \cdot x_3 \cdot y_3^2 + d \cdot x_3^2 + e \cdot y_3^2 + f \cdot x_3 \cdot y_3 + g \cdot x_3 + h \cdot y_3 + j$$

In matrix form this system is written as

$$\begin{bmatrix} x_1^2 \cdot y_1^2 & x_1^2 \cdot y_1 & x_1 \cdot y_1^2 & x_1^2 & y_1^2 & x_1 y_1 & x_1 & y_1 & 1 \\ x_1^2 \cdot y_2^2 & x_1^2 \cdot y_2 & x_1 \cdot y_2^2 & x_1^2 & y_2^2 & x_1 y_2 & x_1 & y_2 & 1 \\ x_1^2 \cdot y_3^2 & x_1^2 \cdot y_3 & x_1 \cdot y_3^2 & x_1^2 & y_3^2 & x_1 y_3 & x_1 & y_3 & 1 \\ x_2^2 \cdot y_1^2 & x_2^2 \cdot y_1 & x_2 \cdot y_1^2 & x_2^2 & y_1^2 & x_2 y_1 & x_2 & y_1 & 1 \\ x_2^2 \cdot y_2^2 & x_2^2 \cdot y_2 & x_2 \cdot y_2^2 & x_2^2 & y_2^2 & x_2 y_2 & x_2 & y_2 & 1 \\ x_2^2 \cdot y_3^2 & x_2^2 \cdot y_3 & x_2 \cdot y_3^2 & x_2^2 & y_3^2 & x_2 y_3 & x_2 & y_3 & 1 \\ x_3^2 \cdot y_1^2 & x_3^2 \cdot y_1 & x_3 \cdot y_1^2 & x_3^2 & y_1^2 & x_3 y_1 & x_3 & y_1 & 1 \\ x_3^2 \cdot y_2^2 & x_3^2 \cdot y_2 & x_3 \cdot y_2^2 & x_3^2 & y_2^2 & x_3 y_2 & x_3 & y_2 & 1 \\ x_3^2 \cdot y_3^2 & x_3^2 \cdot y_3 & x_3 \cdot y_3^2 & x_3^2 & y_3^2 & x_3 y_3 & x_3 & y_3 & 1 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \\ z_8 \\ z_9 \end{bmatrix} \qquad [3]$$

or, in terms of equivalent matrix variables

$$X \cdot a = z \qquad [4]$$

We could solve this system for the coefficients, but that requires inverting the *X* matrix. We can avoid inverting the matrix, and get the solution more quickly, if we scale the x- and y-coordinates such that

$$x_1 \equiv u_1 = 0 \qquad x_2 \equiv u_2 = 1 \qquad x_3 \equiv u_3 = 2 \qquad [5]$$

$$y_1 \equiv v_1 = 0 \qquad y_2 \equiv u_2 = 2 \qquad y_3 \equiv v_3 = 2 \qquad [6]$$

We scale the original *x* and *y* variables (at which to interpolate) with

$$u = \begin{Bmatrix} \frac{x - x_1}{x_2 - x_1} \mid x < x_2 \\ \frac{x - x_2}{x_3 - x_2} + 1 \mid x \geq x_2 \end{Bmatrix} \quad \text{and} \quad v = \begin{Bmatrix} \frac{y - y_1}{y_2 - y_1} \mid y < y_2 \\ \frac{y - y_2}{y_3 - y_2} + 1 \mid y \geq y_2 \end{Bmatrix} \qquad [7]$$

With this scaling, the original 9x9 matrix becomes a matrix of constants, since all the *u* and *v* are constants 0, 1 and 2, regardless of the actual values of *x* and *y*. Since the matrix is constant, its inverse is constant, and we can calculate it once, in advance, and never need to calculate it again.

When we substitute the *u*'s and *v*'s for the *x*'s and *y*'s in the matrix, we have

$$X = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 4 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 4 & 2 & 4 & 1 & 4 & 2 & 1 & 2 & 1 \\ 0 & 0 & 0 & 4 & 0 & 0 & 2 & 0 & 1 \\ 4 & 4 & 2 & 4 & 1 & 2 & 2 & 1 & 1 \\ 16 & 8 & 8 & 4 & 4 & 4 & 2 & 2 & 1 \end{bmatrix} \qquad [8]$$

and the inverse is

$$X^{-1} = \begin{bmatrix} \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} & -\frac{1}{2} & 1 & -\frac{1}{2} & \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \\ -\frac{3}{4} & 1 & -\frac{1}{4} & \frac{3}{2} & -2 & \frac{1}{2} & -\frac{3}{4} & 1 & -\frac{1}{4} \\ -\frac{3}{4} & \frac{3}{2} & -\frac{3}{4} & 1 & -2 & 1 & -\frac{1}{4} & \frac{1}{2} & -\frac{1}{4} \\ \frac{1}{2} & 0 & 0 & -1 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & -1 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{9}{4} & -3 & \frac{3}{4} & -3 & 4 & -1 & \frac{3}{4} & -1 & \frac{1}{4} \\ -\frac{3}{2} & 0 & 0 & 2 & 0 & 0 & -\frac{1}{2} & 0 & 0 \\ -\frac{3}{2} & 2 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad [9]$$

and the coefficient vector solution *a* is found simply by

$$a = X^{-1} \cdot z \quad [10]$$

Again X^{-1} is a constant matrix, so the coefficients for the interpolating polynomial are found with a single matrix multiply. Another advantage to this method is that all of the elements of the inverted matrix can be represented exactly with 89/92+ BCD arithmetic, so they do not contribute to round-off error in the final result

.Once we have the polynomial coefficients, it is straight-forward to interpolate for *z* with

$$z = a \cdot u^2 \cdot v^2 + b \cdot u^2 \cdot v + c \cdot u \cdot v^2 + d \cdot u^2 + e \cdot v^2 + f \cdot u \cdot v + g \cdot u + h \cdot v + i \quad [11]$$

This function, *intrp9z()*, implements these ideas.

```
intrpz9(xl,yl,zmat,x,y)
Func
©({xlist},{ylist},[zmatrix],x,y) 9-point z-interpolation
©Uses matrix math\im1a
©1aprØ1/dburkett@infinet.com

local u,v

when(x<xl[2],(x-xl[1])/(xl[2]-xl[1]),(x-xl[2])/(xl[3]-xl[2])+1)→u

when(y<yl[2],(y-yl[1])/(yl[2]-yl[1]),(y-yl[2])/(yl[3]-yl[2])+1)→v

sum(mat▸list(math\im1a*(augment(augment(zmat[1],zmat[2]),zmat[3]))ᵀ)*{u^2*v^2,u^
2*v,u*v^2,u^2,v^2,u*v,u,v,1})

EndFunc
```

Note that the matrix *im1a* (from equation [9]) must be present and stored in the *\math* folder.

The input arguments are

xl	Three-element list of the table x-coordinates {x1,x2,x3}
yl	Three-element list of the table y-coordinates {y1,y2,y3}
zmat	3x3 matrix of the table z-values
x	The x-value at which to interpolate
y	The y-value at which to interpolate

The z-values are passed as a 3x3 matrix only because that is a convenient way for the user to enter the z-values, directly as they appear in a printed table.

The first two *when()* functions scale *x* and *y* to *u* and *v* as shown in [7]. The last line of the function calculates and returns the interpolation result. The *augment(augment(...))^T* convertst the 3x3 z-matrix to a single-column vector, to perform the matrix multiplication in [10]. The coefficient solution vector is converted to a list, so that the built-in list multiplication can be used to find each term of the sum in [11].

As an example, suppose that we want to interpolate with this table at x = 0.27 and y = 0.55:

	0.4	0.5	0.6
0.1	0.1692	0.2571	0.3616
0.2	0.1987	0.2860	0.3894
0.3	0.2474	0.3335	0.4350

So we have

xl = {.1, .2, .3}
yl = {.4, .5, .6}

$$\text{zmat} = \begin{bmatrix} .1692 & .2571 & .3616 \\ .1987 & .2860 & .3894 \\ .2474 & .3335 & .4350 \end{bmatrix}$$

x = 0.27
y = 0.55

and the call to do the interpolation looks like

```
intrpz9({.1,.2,.3},{.4,.5,.6},[.1692,.2571,.3616;.1987,.286,.3894;.2474,.3335,
.435],0.27,0.55)
```

which returns 0.3664.

This program, *intrp9ui()*, provides a user interface for *intrpz9()*.

```
intrp9ui()
Prgm
©UI for intrpz9()
©calls math\intrpz9, util\rq2v, util\rq3v
©calls util\copyto_h by Samuel Stearly
©31mar01/dburkett@infinet.com

local x123,y123,zz1,zz2,zz3,xy,res,xl,yl,zmat,x,y,z,inhome,errdisp

©Error message display program
define errdisp(msg)=Prgm
 dialog
  title "INPUT ERROR"
  text "Entry error for"
  text msg
  text "Push ENTER to try again"
 enddlog
EndPrgm

©Create lists & matrix
newlist(3)→xl
newlist(3)→yl
```

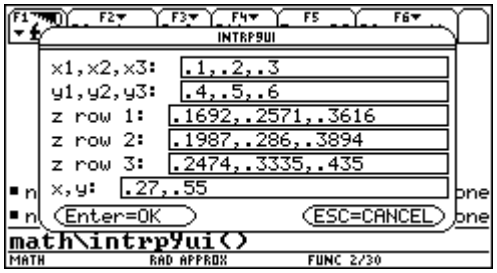
```
newmat(3,3)→zmat

©Initialize variables
"0,0,0"→x123
"0,0,0"→y123
"0,0,0"→zz1
"0,0,0"→zz2
"0,0,0"→zz3
"0,0"→xy
1→inhome

Lbl in1

©Prompt for input variables
dialog
 title "INTRP9UI"
 request "x1,x2,x3",x123
 request "y1,y2,y3",y123
 request "z row 1",zz1
 request "z row 2",zz2
 request "z row 3",zz3
 request "x,y",xy
enddlog
if ok=0:return

©Extract x-variables
util\rq3v(x123)→res
if res="ERR" then
 errdisp("x1,x2,x3")
 if ok=0 then
  return
 else
  goto in1
 endif
else
 res[1]→xl[1]
 res[2]→xl[2]
 res[3]→xl[3]
endif

©Extract y-variables
util\rq3v(y123)→res
if res="ERR" then
 errdisp("y1,y2,y3")
 if ok=0 then
  return
 else
  goto in1
 endif
else
 res[1]→yl[1]
 res[2]→yl[2]
 res[3]→yl[3]
endif

©Extract row 1 z-variables
util\rq3v(zz1)→res
if res="ERR" then
 errdisp("z row 1")
 if ok=0 then
  return
 else
  goto in1
 endif
else
```

```
 res[1]→zmat[1,1]
 res[2]→zmat[1,2]
 res[3]→zmat[1,3]
endif

©Extract row 2 z-variables
util\rq3v(zz2)→res
if res="ERR" then
 errdisp("z row 2")
 if ok=0 then
  return
 else
  goto in1
 endif
else
 res[1]→zmat[2,1]
 res[2]→zmat[2,2]
 res[3]→zmat[2,3]
endif

©Extract row 3 z-variables
util\rq3v(zz3)→res
if res="ERR" then
 errdisp("z row 3")
 if ok=0 then
  return
 else
  goto in1
 endif
else
 res[1]→zmat[3,1]
 res[2]→zmat[3,2]
 res[3]→zmat[3,3]
endif

©Extract x,y variables
util\rq2v(xy)→res
if res="ERR" then
 errdisp("x,y")
 if ok=0 then
  return
 else
  goto in1
 endif
else
 res[1]→x
 res[2]→y
endif

©Do the interpolation
math\intrpz9(xl,yl,zmat,x,y)→z

©Display result
dialog
 title "RESULT"
 text "x: "&string(x)
 text "y: "&string(y)
 text ""
 text "z = "&string(z)
 text ""
 dropdown "Put z in Home?",{"no","yes"},inhome
 text ""
 text "Push ENTER to interpolate again"
enddlog
```

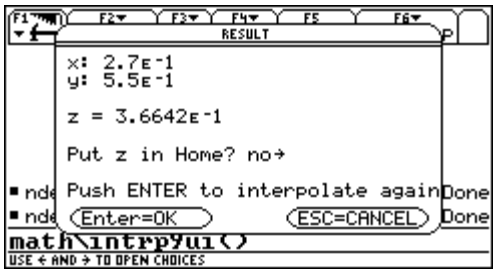
```
©Copy result to home screen
if inhome=2
util\copyto_h(string(x)&","&string(y),z)

©Exit, or interpolate again
if ok=0: return
goto in1

EndPrgm
```

Refer to the comments at the start of the program: it requires that a few functions be installed in the *\math* and *\util* folders.

When the program is run, an input screen is shown. This screen shot shows the values entered for the example above.

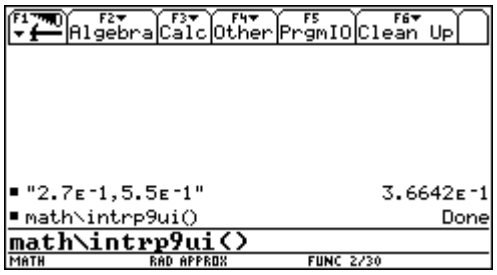


You can push [ESC] to exit the program and return to the home screen. Note that several parameters are entered in each entry field. When all the parameters are entered, the solution is calculated and displayed in this result screen:



You can press [ENTER] to return to the input screen and do another interpolation, or press [ESC] to quit the program and return to the home screen.

For each interpolation, you can copy the interpolation result to the home screen by selecting YES in the drop-down menu labelled *Put z in Home?*. If you choose this option, the results are copied to the home screen when you exit the program. For the example, the home screen would look like



The result is shown in the second level of the history display. The left-hand entry shows the x- and y-coordinates as a string, and the right-hand entry is the interpolated result. This feature is made possible by Samuel Stearly's *copyto_h()* function, as described in tip [7.8].

If you forget to enter one of the commas that separate the arguments, a dialog box error message is shown:

This example shows that an error has been made in entering the *y1,y2,y3* arguments. When [ENTER] is pressed, the input screen is again shown, and you can correct the error.