Fitting Implicit Functions

Occasionally you may need to fit data to a model that doesn't have a form that can be expressed as *y*=f(*x*). An example would be fitting a circle or ellipse using an equation like

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

Because this equation can't be expressed as *y*=f(*x*), you can't write a standard user-defined fitting function for it.

This problem is related to the errors in variables problem in which data has measurement errors in both the dependent and independent variables. You have two inputs, or independent variables (*x* and *y*), both with (probably) similar measurement errors. It differs from errors in variables fitting in that the function output is zero instead of being a dependent variable.

The ODRPACK95 package also supports fitting to implicit functions, which you can do with the /ODR=3 flag. You create a fitting function with multiple independent variables (in the case of the ellipse above, two for *x* and *y*). The fitting process will attempt to find *x* and *y* residuals and fit coefficients to minimize the distance from the data to the zero contour of the function.

It may be a good idea at this point to read the section about errors in variables fitting; see **Errors in Variables: Orthogonal Distance Regression** on page III-236; much of it applies also to implicit fits.

There are a few differences between regular fitting and implicit fitting. For implicit fits, there is no autodestination, and the report printed in the history does not include the wave assignment showing how to get values of the fitted model (doing that is actually not at all easy).

Because of the details of the way ODRPACK95 operates, when you do an implicit fit, the curve fit progress window does not update, and it appears that the fit takes just one iteration. That is because all the action takes place inside the call to ODRPACK95.

The fit function must be written to return zero when the function solution is found. So if you have an equation of the form f(x_i) = 0, you are ready to go; you simply create a fit function that implements f(x_i). If it is in the form f(x_i) = constant, you must move the constant to the other side of the equation: f(x_i)-constant = 0. If it is a form like f(x_i) = g(x_i), you must write your fitting function to return f(x_i) - g(x_i).

Example: Fit to an Ellipse

In this example we will show how to fit an equation like the one above. In the example the center of the ellipse will be allowed to be at nonzero x_0, y_0.

First, we must define a fitting function. The function includes special comments to get mnemonic names for the fit coefficients (see **The New Fit Function Dialog Adds Special Comments** on page III-253). To try the example, you will need to copy this function and paste it into the Procedure window:

```
Function FitEllipse(w,x,y) : FitFunc
    Wave w
    Variable x
    Variable y

    //CurveFitDialog/
    //CurveFitDialog/ Coefficients 4
    //CurveFitDialog/ w[0] = a
    //CurveFitDialog/ w[1] = b
    //CurveFitDialog/ w[2] = x0
    //CurveFitDialog/ w[3] = y0

    return ((x-w[2])/w[0])^2 + ((y-w[3])/w[1])^2 - 1
End
```

An implicit fit seeks adjustments to the input data and fit coefficients that cause the fit function to return zero. To implement the ellipse function above, it is necessary to subtract 1.0 to account for "= 1" on the right in the equation above.

The hard part of creating an example is creating fake data that falls on an ellipse. We will use the standard parametric equations to do the job (y = a*cos(theta), x=b*sin(theta)). So far, we will not add "measurement error" to the data so that you can see the ellipse clearly on a graph:

```
Make/N=20 theta,ellipseY,ellipseX
theta = 2*pi*p/20
ellipseY = 2*cos(theta)+2
ellipseX=3*sin(theta)+1
```

A graph of the data (you could use the Windows→New Graph menu, and then the Modify Trace Appearance dialog):

```
Display ellipseY vs ellipseX
ModifyGraph mode=3,marker=8
ModifyGraph width={perUnit,72,bottom},height={perUnit,72,left}
```

The last command line sets the width and height modes of the graph so that the ellipse is shown in its true aspect ratio. Now add some "measurement error" to the data:

```
SetRandomSeed 0.5       // so that the "random" data will always be the same…
ellipseY += gnoise(.3)
ellipseX += gnoise(.3)
```

Now you can see why we didn't do that before — it's a pretty lousy ellipse!

Now, finally, do the fit. That requires making a coefficient wave and filling it with the initial guesses, and making a pair of waves to receive estimated values of X and Y at the fit:

```
Duplicate ellipseY, ellipseYFit, ellipseXFit
Make/D/O ellipseCoefs={3,2,1,2}                    // a, b, x0, y0
FuncFit/ODR=3 FitEllipse, ellipseCoefs /X={ellipseX, ellipseY} /XD={ellipseXFit,ellipseYFit}
```

The call to the FuncFit operation has no Y wave specified (it would ordinarily go right after the coefficient wave, ellipseCoefs) because this is an implicit fit.

The results:

```
Fit converged properly
  ellipseCoefs={3.1398,1.9045,0.92088,1.9971}
  V_chisq= 1.74088; V_npnts= 20; V_numNaNs= 0; V_numINFs= 0;
  W_sigma={0.158,0.118,0.128,0.0906}
  Coefficient values ± one standard deviation
   a   =3.1398 ± 0.158
   b   =1.9045 ± 0.118
   x0  =0.92088 ± 0.128
   y0  =1.9971 ± 0.0906
```

And add the destination waves (the ones specified with the /XD flag) to the graph:

```
AppendToGraph ellipseYFit vs ellipseXFit
ModifyGraph mode=3,marker(ellipseYFit)=1
ModifyGraph rgb=(1,4,52428)
```