
Subject: efficient polynomial fitting

Posted by [mperrin+news](#) on Thu, 16 Oct 2003 06:50:17 GMT

[View Forum Message](#) <> [Reply to Message](#)

What's the most efficient way to fit a polynomial of low order (~ 3) to several hundred thousand data points? I feel there has to be a more IDL-ish, matrix based solution than a for loop calling poly_fit 300,000 times (which is of course sloooooow).

My first approach was to code up a straightforward linear-least-squares polynomial fitter and append an extra dimension to everything to do the fit for all my points in parallel, but this fails, due to the behavior of the matrix multiply operator.

Let "deg" be the degree of the polynomial I'm trying to fit, and let's consider my data to be an array of size [nt,nx]. I want to do the polynomial fit over the t dimension, while the points are completely independent in the x direction. That is, I'm **not** trying to do a two-D least squares - I just want to do least squares on tons of point simultaneously. So here goes:

```
; basic setup for least squares fit, with the nx dimension tacked on...
pow = indgen(deg+1)
powarr = rebin(transpose(pow),nt,deg+1,nx)

ti = rebin(reform(double(data),nt,1,nx),nt,deg+1,nx)
tarr = t^powarr
tarr_transpose = transpose(tarr,[1,0,2])

; here's where we get into trouble
alpha = tarr ## tarr_transpose ; alpha should be (d+1)*(d+1)*nx, but is just (d+1)*(d+1)
```

In other words, I want to matrix multiply a [nt,d+1,nx] * [d+1,nt,dx] and only have the multiply operate over the first two dimensions, leaving the third alone. Is there any way to do this in IDL, **without** resorting to a for loop of some kind?

- Marshall
