Subject: Re: Simultaneous fitting in IDL
Posted by Allan Whiteford on Mon, 18 May 2009 16:27:16 GMT
View Forum Message <> Reply to Message

Gianluca,

Gianluca Li Causi wrote:
> On May 15, 1:15 pm, Allan Whiteford
> <allan.rem...@phys.remove.strath.ac.remove.uk> wrote:
>> Gianluca,
>>
>> If you really believe your data and, more importantly, really believe
>> the error bars associated with them then it's formally correct that all
>> of your Y2 data points are essentially being ignored. Bet you're sorry
>> you went to the trouble of measuring them now? :)
>
>
> Dear Allan,
> thank you much for your reply, but I do not fully agree qith you.
>
> A can assume that both my Y1 and Y2 data are random and not
> correlated,

I have no idea where your data are coming so you are the best person to say how valid that assumption is - it's not typical for that assumption to be valid (but it is typical for that assumption to be made).

> but they are not equally dependent on the parameters P=
> [P1,P2,P3]: in fact the F1 function is weakly dependent on the last
> parameter P3, while the F2 function is heavily dependent on it.
>

The key thing here is lots of points with a weak dependence and a few points with a strong dependence. This is very different from zero dependence.

> So, if the Y2 data are essentially ingnored I get a large
> indetermination of P3, although the Y2 data are enough to constraint
> it very well!
> Instead if I modify the weight of Y2, i.e. its error, as I do, I get a
> nice result for any of my parameters, expecially the P3 given that the
> F2 function passes very well across the Y2 data points!
>

You have a pre-disposition to want the two curves to fit equally well. Your fitting algorithm has a pre-disposition to want every point to fit equally well (within its error bars).

> At the limit where each function F_i only depends on a single
> parameter P_i, the simultaneous fitting of the joined Y_i vectors
> shoud give the same result of the independent one-parameter fitting of
> each Y_i dataset, shoudn't it?

Yes.

> My procedure seems to satisfy this limit, which it seems to me is not
> satisfied by your suggestion to leave the errors as they are.
>

You'll end up inverting a block diagonal matrix which is simply the
inverse of the two blocks pushed together - this will satisfy your
limiting case (matrix inversion stability notwithstanding).

>
>
>> The simple scenario when they wouldn't be ignored would be when one (or
>> more) of the parameters only has a significant effect on the modelling
>> of Y2 data. This doesn't seem to be the case here since when you mess
>> with the errors you do get a change in your fit.
>>
>> Are the errors on your points (particularly your Y1 points) truly
>> random? Chances are your Y1 error estimates don't contain only random
>> errors but also have some form of systematic error - hence correlation
>> between the points which is a whole new can of worms.
>>
>> In the case of correlated errors, your weight vector needs to become a
>> matrix with the diagonal elements being your weight as before and the
>> off-diagonals being the correlation between points (you'll need to
>> invert this matrix along the way - hopefully it's not too big). When you
>> include this (assuming it has off-diagonal elements which are
>> significant) one of the results will be that more attention is paid to
>> Y2 data making you glad you went to the trouble of measuring them :).
>
> I agree with you in this case, when the errors are not truly random,
> but are correlated: do you know hoe to use the LMFIT routine in this
> case? How can I pass to LMFIT the errors correlation matrix in place
> of the Y_err vector?
>

I've never used LMFIT. I'm only assuming LM stands for
Levenberg-Marquadt. Actually, see below where I use it for the first
time. It doesn't look like it will do correlated errors though.

>
>
>> Scientists typically compensate for this whole complicated correlation

>> problem by messing around with weighting of the errors in a similar way
>> to what you have been trying - it can often work quite well. Sometimes
>> they realise what they are compensating for but mostly they just do it
>> because they get curves which look nicer and they have a gut feeling
>> that some measurements have to count for something.
>>
>> After you've messed with your errors/weights though, don't expect formal
>> statistical tests to have much meaning. Your problem in this case is
>> that your chi2 statistic is still behaving correctly and saying "most of
>> these points are further away than they should be" because you've
>> mangled the fit so that it moves Y1 points away to compensate for Y2
>> points. Who knows what the errors in your measured parameters now
>> represent... but they won't be correct!
>
> Of course, the best would be to have an LMFIT routine which accepts
> more than one dataset and computes internally the Reduced ChiSquare
> for each of them and the Total Reduced ChiSquare to be minimized as
> the average of the single Reduced ChiSquares...
>
> Does anybody have a multiple-fitting LMFIT routine?
>

It's formally the same thing as you are doing. The algorithm doesn't
care where your data come from.

> By the way, I think I could use the same limit told before, to check
> if in this limit I get the same parameters error as in the independent
> one-parameter fittings...
> This maybe should prove once for all what of our tuw approach is the
> right one!
>

The following code does this, more or less. The first 1000 points only
depend on p[0] and p[1] while the last 10 only depend on p[2]. Fitting
together or separately gives the same parameter estimates and errors on
them:

```
function one,x,p
 return,[p[0]+p[1]*x,1.0,x]
end

function two,x,p
 return,[p*x^2,x^2]
end

function combined,x,p
 if x lt 1000 then begin
        return,[one(x,p[0:1]),0.0]
```

```
        endif else begin
         tmp=two(x,p[2])
         return,[tmp[0],0.0,0.0,tmp[1]]
        endelse
end

; Initial parameters
p=[4.,5.,7.]
x=findgen(1010)

; Make up some data and errors
data=x*0.0
for i=0,n_elements(x)-1 do data[i]=(combined(x[i],p))[0]
data=data+sqrt(data)*(randomu(seed)-0.5)
error=sqrt(data)

; Combined fit
p=[3.8,5.1,7.1]
hmm=lmfit(x,data,p,function_name='combined',sigma=sigma, $
        measure_errors=error,/double)
print,p,sigma

; First two parameters fit
p=[3.8,5.1]
hmm=lmfit(x[0:999],data[0:999],p,function_name='one', $
        sigma=sigma,measure_errors=error[0:999],/double)
print,p,sigma

; Last parameter fit
p=[7.1]
hmm=lmfit(x[1000:*],data[1000:*],p,function_name='two', $
        sigma=sigma,measure_errors=error[1000:*],/double)
print,p,sigma

end
```

As you can see the two independent methods give the same results (and, importantly, the same error estimations in the fitted variables). The key thing is that here there is zero dependence whereas you have weak dependence.

Note that I made up the line about "data=data+sqrt(data)*(randomu(seed)-0.5)" - it's probably nonsense but the key thing is we get the same results by the two different methods.

(Written in a hurry - apologies for any mistakes.)

Thanks,

Allan

> What do you think about?
> Gianluca
>
>