Subject: Re: Generalized Least Squares? (LONG POST!)
Posted by Craig Markwardt on Tue, 20 May 2008 07:06:24 GMT
View Forum Message <> Reply to Message

Gernot Hassenpflug <gernot@nict.go.jp> writes:
> Dear all,
>
> I'm involved in ongoing research on a problem that I solved to a first
> approximation with weighted least squares (using MPFIT) but for a real
> solution I require generalized least squares: WLS uses a diagonal
> covariance matrix, i.e., the data errors are uncorrelated; GLS uses a
> full covariance matrix, i.e., the data errors can be correlated.
>
> I have not found any ready solution in IDL yet, and I am under the
> impression that there is no analytical solution to GLS, so fairly
> complicated numerical methods are required.
>
> I have actually found a routine in MATLAB called "lscov" which can
> solve this problem, and wonder whether there is a chance to hobble
> something together in IDL? I'd be happy to try and modify MPFIT to be
> able to deal with GLS too, could anyone comment on possibilities?

Greetings Gernot--

Solving the generalized least squares problem is reasonably straightforward. Basically it involves transforming the original (correlated) variables to new (uncorrelated) variables. Ironically I did this all in C instead of IDL because of project requirements, but the concepts are the same.

The least-squares problem can be expressed as the following normal equation, in matrix notation,

A^T A x = A^T v

where A is the pattern matrix (= "Jacobian"), v is the vector of normalized residuals, and x is the vector of parameters (and "^T" indicates the matrix transpose). MPFIT solves this equation, given an initial estimate of x, to get a new estimate of x.

I say that v is the vector of normalized residuals, because typically we compute it something like this,

v = (DATA - MODEL)/ERROR

This explicit definition shows that we were assuming *uncorrelated* errors, i.e. for each vector element, there is a single well defined uncertainty which does not depend on neighboring elements. The

chi-squared value is defined as,

  CHI2 = v^T v

However, in the case of correlated errors, the chi-square value is defined as,

  CHI2 = v^T (COVAR^(-1)) v

where COVAR is the covariance matrix and "^(-1)" indicates the matrix inverse. In this case, v is no longer the normalized residuals, but just the raw residuals. The units are correct since COVAR has units of v^2, so (COVAR^(-1)) has units of v^(-2). The normal equation becomes,

 A^T (COVAR^(-1)) A x = A^T (COVAR^(-1)) v

Unfortunately, MPFIT does *not* solve this problem. Are we out of luck? No, actually it's still a reasonably straightforward problem to solve.

For example, consider if we can factor the COVAR matrix like this,

 COVAR = L L^T

where L is lower-triangular. This is the well-known Cholesky factorization. As long as COVAR is positive-definite, as it should be, the Cholesky factorization is well-defined, and of course there is a routine to perform this factorization within IDL, using the procedure CHOLDC. (more on this below),

In that case, it's possible to re-write the normal equations as,

 B^T B x = B^T u

where
 B = (L^(-1)) A
 u = (L^(-1)) v

MPFIT *can* solve the B equation. This is effectively a new problem, but with pattern matrix B and residuals u that have their correlations removed, compared to the original A and v matrices. The B and u equations can be re-written as,

 L B = A
 L u = v

where are immediately solvable by IDL using CHOLSOL. Actually, the

first equation is a stack of N equations, but it can be solved by
calling CHOLSOL N times.

OK, so how does this practically work within IDL? Well, let's assume
that we start with a function MYFUNCT which computes the unweighted
residuals and Jacobian,

```
FUNCTION MYFUNCT, p, dp, _EXTRA=extra
 ...
END
```

We can create a new (untested!) function, MYFUNCT_CORREL, which
handles the covariance, like this,

```
FUNCTION MYFUNCT_CORREL, p, dp, COVAR=covar,_EXTRA=extra
  ;; If Jacobian is requested
  if n_params() GT 1 AND n_elements(dp) GT 0 then dp0 = dp

  ;; Raw residuals V and Jacobian DP from the original function
  v0 = MYFUNCT(p, dp0, _EXTRA=extra)

  ;; Correct for correlations
  L = CHOLDC(covar, LDIAG)
  v = CHOLSOL(L, LDIAG, v0)  ;; Residuals
  if n_elements(dp0) GT 0 then begin
    ;; Jacobian matrix
    dp = dp0
    for i = 0, n_elements(p)-1 do dp(*,i) = CHOLSOL(L, LDIAG, dp0(*,i))
  endif

  ;; Return the corrected residuals, and (implicitly) the Jacobian
  return, v
END
```

One thing you will see is that L, the solution computed by CHOLDC(),
doesn't change, so you could optimize by computing this value only
once. Actually, MPFIT already has an (undocumented) keyword called
SCALE_FCN which could much of the work done by MYFUNCT_CORREL. It's a
user-function which accepts a residual vector V and Jacobian matrix DP
and modifies them according to the above prescription.

OK, this all may sound great. Unfortunately, in my particular
application, I did not succeed. The problem was that my particular
covariance matrix was not positive-definite. I had huge correlations
between points, which caused the CHOLSOL stage to fail.

There is theoretically a way around *this* problem as well. Instead
of using the Cholesky factorization, one can use the SVD

factorization, which is far more robust against singular matrices.
The SVD factorization looks like this,

  COVAR = UMAT WMAT VMAT^T

where UMAT, VMAT and WMAT are matrices with special properties. In IDL, the SVDC procedure computes this factorization.

The benefit of this method is that the singular values are sorted by magnitude within the WMAT matrix. The first values are the strongest, and the last values are small, or zero. One can effectively "zero out" the insignificant singular values, which results in a more robust effective inverse, COVAR^(-1). This is described in more detail by Numerical Recipes. However, a more intuitive way to think about this is that if you start with N measurements, but M of the singular values are insignificant, then your data set really had N-M uncorrelated degrees of freedom to begin with (whereas M of the measurements were effectively totally dependent values).

Proceeding, one can compute the revised values,

  B = (WMAT^(-1/2)) VMAT^T A
  u = (WMAT^(-1/2)) VMAT^T v

and then the problem is reduced again to the "uncorrelated" normal equations described earlier. Since WMAT is a diagonal matrix, all of the equations above are very easy to compute, and can be substituted into MYFUNCT_CORREL.

Again, for my problem, I implemented this method in the C language, but to be honest, the method did not improve the situation. I believe that my data was so correlated that even SVD was not appropriate. I eventually I gave up to work on other things. However, in principle this solution is correct.

Hope this was helpful!
Craig
