Subject: !ERR and MPFIT
Posted by Craig Markwardt on Tue, 16 Nov 1999 08:00:00 GMT
View Forum Message <> Reply to Message

When I set out to translate MPFIT from MINPACK, I wanted to stay as faithful as possible, but I also wanted it to be as simple as possible. When it came to error handling, I wanted the user's model routine to be able to report an error condition.

Unfortunately I chose to use the !ERR system variable. If !ERR is set to a negative number, then MPFIT aborts the run, assuming that there was an unrecoverable error.

I now realize that using !ERR was probably a mistake. Why? RSI seems to want to make it obsolete. Through their error-handling flavor of the month program, !ERR seems to have fallen out of favor. Also, there are quite a few actions which might set !ERR accidentally in the user's function without actually signalling an error condition.

So I have two questions:

* to people who use MPFIT: does anybody actually use the !ERR status variable to control the fitting process? If not, then I would consider removing it.

* to everybody: any suggestions on how to generically signal an error condition? My thoughts were:

  - ERROR keyword variable - don't like this, since then the function has to accept keywords

  - define a new system variable - don't like this either, since it's hard to do system variables right

  - common block variable - not very pretty, but gets the job done.

To be clear, this is some kind of error flag that a user routine would (optionally!) set to signal an abnormal termination condition. Right now I am leaning toward the common-block approach. Sorry David.

Thanks,

Craig

--
---------- ----------
Craig B. Markwardt, Ph.D. EMAIL: craigmnet@cow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response

---------- ----------

Subject: Re: !ERR and MPFIT
Posted by R.Bauer on Sun, 28 Nov 1999 08:00:00 GMT
View Forum Message <> Reply to Message

Craig Markwardt wrote:

> m218003@modell3.dkrz.de (Martin Schultz) writes:
>>
>> In article <MPG.129be687cb5374e598996c@news.frii.com>,
>> davidf@dfanning.com (David Fanning) writes:
>>> Craig Markwardt (craigmnet@cow.physics.wisc.edu) writes:
>>>
>>>> now I am leaning toward the common-block approach. Sorry David.
>>>
>>> Oh, I like it. And to tell you the truth, this might
>>> be the *perfect* situation for a common block. Just
>>> don't be putting 'em in a widget program! :-)
>>>
>>
>> Now, what happens if you have two or three widgets open each of which
>> is calling MPFIT through some means. Would a common block still work?
>> And please think once more: may not be possible now, but I am quite certain
>> that RSI will one day support SMP, so it could indeed happen that those
>> calls to MPFIT were executed simultaneously! I'd go for the keyword - this
>> is clean.
>
> I totally understand what you are saying. Keywords make everything
> clean. But consider the following function:
>
> function myfunc, x, p
> ...
> end
>

You should have a look at

_REF_EXTRA: Passing Keyword Parameters by Reference

You can pass keyword parameters to called routines by reference by adding the formal keyword parameter "_REF_EXTRA" (note the underscore character) to the definition of your routine. Passing parameters by reference means that you are giving the called routine the name of an existing IDL variable to work with; IDL takes care of keeping track of the value associated with the name. The values of keyword parameters specified via _REF_EXTRA are not available to the routine that is passing the keywords on.

When a routine is defined with the formal keyword parameter _REF_EXTRA, pairs of unrecognized keywords and values are placed in a storage location that is accessible to both calling and called routines, and the keyword names are placed in an IDL string array. The string array can be "deciphered" using the _EXTRA keyword, which matches the names in the string with the "live" values in the storage location. This means that if the keywords specify IDL variables, the values of those variables can be altered by any routine that has access to the variable via the keyword inheritance mechanism. In this fashion, the values of keyword parameters can be changed within a routine and passed back to the routine's caller.

The "pass by reference" keyword inheritance mechanism is especially useful when writing object methods, which may be inherited multiple times and which often wish to change the value of variables available to the calling method. (The values of object properties are one example of data that can profitably be shared by objects at various levels in an object hierarchy.)

>
> It doesn't accept keywords. Now, if MPFIT tries to call this function
> with an ERROR keyword, everything crashes. I could try CATCHing such
> an error, and retrying the function call with without the ERROR
> keyword, but then what's more ugly? By the way, I've changed the
> implementation of MPFITFUN to use common blocks instead of pointers
> (handles really), and it's become much more clean and easy to read
> now.
>

I am myself don't like common blocks. I prefer _extra and _ref_extra or heap variables

>
> The common block implementation has its virtues. It's totally
> optional. It wouldn't collide with any other system error variables.
> It's certainly better than my current use of !ERR. And, currently,
> it's guaranteed that only one session of MPFIT can be running
> simultaneously.

> I am sure that when (if!) RSI implements multithreaded IDL, almost

> everything is going to crash. Not just MPFIT. Consider every program
> that uses common blocks, *especially* the thousands of IDL library
> scripts, assumes a single thread of execution. RSI will have to
> implement some kind of new functionality to keep things working and I
> for one will depend on that! :-)

I hope that's common blocks will be obsolete in future. So everybody - RSI too will clean up those programs.

Reimar
