Subject: Re: Specification for a new array slicing function
Posted by steinhh on Thu, 20 May 1999 07:00:00 GMT
View Forum Message <> Reply to Message

In article <374317CC.E1AC89EA@ssec.wisc.edu> Liam Gumley <Liam.Gumley@ssec.wisc.edu> writes:

```
>  Please find below a suggested specification for a new array slicing
>  function, formatted as a standard IDL prolog. The intention here is to
>  provide a means to extract n-dimensional array 'slices' from an existing
>  array in memory. The caller can choose to skip elements along any or all
>  dimensions if desired.
>
>  Comments are invited. There's no code yet, so now is the time.
>
[..]
>  ; INPUT KEYWORD PARAMETERS:
>  ;    START       Set this keyword to a vector containing the start
[..]
>  ;    STRIDE      Set this keyword to a vector containing the
[..]
>  ;    COUNT       Set this keyword to a vector containing the
[..]
>  ; EXAMPLE:
>  ;
>  ; ;Extract every other element along each dimension
>  ;
>  ; array = findgen( 1, 10, 5, 6, 7 )
>  ; ndims = size( array, /n_dimensions )
>  ; stride = replicate( 2L, ndims )
>  ; result = array_slice( array, stride=stride )
>  ; help, result
>  ;
>  ; ;RESULT         FLOAT     = Array[1, 5, 2, 3, 3]
```

IMO, the use of keyword parameters for START, STRIDE and
COUNT is a bit "wordy" for my liking. And these items
are really essential to the routine as such. So why not
use positional parameters?

For something that really ought to be a part of the IDL
syntax, I would also like a shorter name (despite the
possibility for name conflicts), like "arex", short
for array_extract.

My suggestion would be something a bit more like the native
Fortran 9X syntax (not that I actually *know* exactly how that
syntax works!) , e.g.:

a(0:5:2,:,5:9) would be translated into

```
  arex(a,[0,5,2],-1,[5,9])
```

I.e., each positional parameter signifies the extraction
operation for one array dimension. There are some issues
that I would like to clear up, though: What exactly does
the 0:5:2 sequence mean? Does it mean elements 0:5, sampled
with a stride of 2? Or does it mean 5 elements sampled with
a stride of 2, starting from 0? Or is it START:STRIDE:COUNT,
meaning 2 elements, sampled with a stride of 5?

Just curious.... And I would strongly recommend following
Fortran conventions, whatever they are....

Anyway, the three elements in each parameter appear in
"optionality" order: start [, stride [, count]] (if that's
what the syntax is supposed to be).

Looking at the example above, you may wonder what the "-1" is
doing there... Well, the idea is that one could use a
nonnegative *scalar* parameter to signify extraction of a
slice at a given position, whilst -1 really means "*", in IDL
notation.

Since we now have one extra "degree of freedom" in that a
start position (and nothing else) may be specified in two
similar ways, as e.g. 0 or [0]... I have great use for
this (since we're at now at liberty to rewrite the rules.. :-)
I've always disliked the way this works:

```
   a = fltarr(5,5,5)
   surface,a(*,3,*)
% SURFACE: Array must have 2 dimensions: <FLOAT     Array[5, 1, 5]>.
```

I mean - if I'm extracting an "image" out of a "cube", why
would I want the last dimension to stick around...???

So, I would like to be able to say

```
   surface,arex(a,-1,3,-1)
```

with no error messages! On the other hand, if I do want the
dangling dimension, I could specify it:

```
   surface,arex(a,-1,[3],-1)
```

(and get the error message :-)

Or the other way around, if people feel strongly about leaving
this dimension in.....

I would also like to see a corresponding index function,
returning the one-dimensional indices to the extracted
elements instead of the elements themselves. This could
be used for assignments. I.e.:

```
   a(arexi(a,-1,[3],[0,2])) = data_block
```

Just some thoughts...

Regards,

Stein Vidar
