Subject: Specification for a new array slicing function
Posted by Liam Gumley on Wed, 19 May 1999 07:00:00 GMT
View Forum Message <> Reply to Message

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.

Cheers,
Liam.

```
;+
; NAME:
;    ARRAY_SLICE
;
; PURPOSE:
;    Extract an n-dimensional slice from an array in memory.
;
; CATEGORY:
;    Array processing.
;
; CALLING SEQUENCE:
;    RESULT = ARRAY_SLICE( ARRAY )
;
; INPUTS:
;    ARRAY       The array from which data will be extracted.
;                ARRAY must be defined and have one or more
;                dimensions, otherwise execution will halt.
;
; OPTIONAL INPUTS:
;    None.
;
; INPUT KEYWORD PARAMETERS:
;    START       Set this keyword to a vector containing the start
;                position for extraction along each dimension
;                (default is [0,0,...,0]).
;                START must have the same number of dimensions
;                as ARRAY, otherwise execution will halt.
;                START is automatically limited to minimum and
;                maximum values suitable for ARRAY.
;    STRIDE      Set this keyword to a vector containing the
;                sampling interval along each dimension
;                (default is [1,1,...1] for contiguous extraction).
;                STRIDE must have the same number of dimensions
```

```
;               as ARRAY, otherwise execution will halt.
;               STRIDE is automatically limited to minimum and
;               maximum values suitable for ARRAY.
;   COUNT       Set this keyword to a vector containing the
;               number of items to extract along each dimension
;               (default is to extract all data).
;               COUNT must have the same number of dimensions
;               as ARRAY, otherwise execution will halt.
;               COUNT is automatically limited to minimum and
;               maximum values suitable for ARRAY.
;
; OUTPUT KEYWORD PARAMETERS:
;   None.
;
; OUTPUTS:
;   RESULT      The extracted array (all dimensions are left intact).
;               If none of START, STRIDE, COUNT are specified,
;               returns a copy of the input array.
;
; OPTIONAL OUTPUTS:
;   None.
;
; COMMON BLOCKS:
;   None.
;
; SIDE EFFECTS:
;   None.
;
; RESTRICTIONS:
;   None.
;
; 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]
;
; MODIFICATION HISTORY:
; $Id: array_slice.pro,v 1.1 1999/05/19 19:44:27 gumley Exp $
;-

---
```

Liam E. Gumley
Space Science and Engineering Center, UW-Madison
http://cimss.ssec.wisc.edu/~gumley

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

In article <7i0igd$1er$1@readme.uio.no>, steinhh@ulrik.uio.no (Stein Vidar Hagfors Haugan) wrote:

> 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....

I have the Numerical Recipies in Fortran 90 book, which has a nice, short introduction to F90 concepts.

The F90 syntax is a(lower:upper:stride) or as I prefer to think about it a(from:to:by). It works exactly like a DO (or FOR) loop, which the new indexing largely replaces. You can have negative strides if upper<lower. If upper-lower has a different sign than stride, you get a null result.

So in F90

b = a(10:1:-1)

would give a in reverse order. I find thi more appealing that having to look up the direction parameters in ROTATE.

In IDL this could be

b = a[*:0L:-1]

Still better than F90 because of the * (an the zero-based indexing, of course :-)).

Finally, much of F90 was developed to make parallel programming easier. I would hope that true parallelism is a goal for IDL (rather than just better ActiveX controls).

Ken

--
Dr. Kenneth P. Bowman, Professor 409-862-4060
Department of Meteorology 409-862-4466 fax
Texas A&M University bowmanATcsrp.tamu.edu
College Station, TX 77843-3150 Replace AT with @

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

Liam Gumley wrote:
>

> I agree that START, STRIDE, COUNT are somewhat wordy. However I'd like
> to be able to specify them in any combination, e.g.
> I'm not sure I know a clean way to allow these combinations, other than
> using optional keywords.

well, if you set the ones you don't need to -1 (as this was suggested to
specify "all"). But at least for STRIDE, -1 should be possible as such
(see my other post).

> START only, or ARREX(array,START)
> STRIDE only, or ARREX(array,-1,-1,STRIDE)
> COUNT only, or ARREX(array,0,COUNT-1) ; you need a start to count !
> START and STRIDE, or ARREX(array,START,-1,STRIDE)
> START and COUNT, or ARREX(array,START,START+COUNT)
> STRIDE and COUNT, or ARREX(array,0,COUNT-1,STRIDE)
> START and STRIDE and COUNT. ARREX(array,START,START+COUNT,STRIDE)

Note that I used START,END,STRIDE as in F90.

Also: while START,END,STRIDE can be multi-dimensional, they must have
the same dimensions - -1 being an exception, e.g.
array=fltarr(10,10,10)
start=[0,0,5]
stride=[2,2,1]
help,arrex(array,start,-1,stride) should yield ARRAY[5,5,5]

Also, perhaps, an undefined parameter should also be interpreted as
"ALL" (and then be returned as -1), i.e. in the above example
arrex(array,start,theend,stride) should yield the same result
although THEEND wasn't defined

A boolean /REFORM keyword would be a nice feature.

Martin.

--

||||||||||||||\\\\\\\\\\\\\\----------////////////// //||||||||||||||
Martin Schultz, DEAS, Harvard University, 29 Oxford St., Pierce 109,
Cambridge, MA 02138 phone (617) 496 8318 fax (617) 495 4551
e-mail mgs@io.harvard.edu web http://www-as/people/staff/mgs/

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

> Stein Vidar Hagfors Haugan wrote:

>> 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])

The Fortran standard is [Start]:[End][:Stride]. ":" without
preceding or following numbers means all elements; any or
all of Start, End, or :Stride can be left off and would default
to first element, last element, :1, respectively.

I don't see why a new function, whatever it's called, should
be needed to handle this. It looks (from the user perspective)
like a simple extension of the current IDL syntax for array
sectioning. And, again from at least this user's perspective,
Sect=a[0:5:2,:,5:9] is preferable to Sect=AS(a,[0,5,2],-1,[5,9]),
which is nowhere near as easy to remember or obvious-looking.

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

Stein Vidar Hagfors Haugan wrote:
>
> 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,
> [...]
>> ; 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.

good points. Although I would add one more character and name it "arrex"
to avoid confusion with "ar"gument or "ar"ea etc. (only "arr"ow left
then ;-)

>
> 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])
>
Sounds nice, however, this is truely up to RSInc to implement. I assume
Liam's proposal was something we, the community, could do ourselves.

Anyway, I asked our Fortran 90 expert, and he told me the following:
- the 3 optional parameters work exactly like a DO (IDL=FOR) loop, i.e.
you have start, end , stride
- you can leave any of them empty which is implicitely defaulted to all,
all, 1
- a statement like A(::1,LM) = A(::LM,1,-1) reverses the last
dimension

If RSInc would go for this, I think they should try to use the same
conventions. It's already bad enough to have to rethink DO and FOR each
time you change.

> 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.

Then, why shouldn't it be "*" as always ?
(or even better, allow the empty field as in F90: A[:1:-1] would be identical to
reverse(A[1:*]) in the current syntax)

>
> 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)
>

this seems to be somewhat messy: the "syntax" would rather be
ARRAY[start1:end1:stride1, start2:end2:stride2, ... , start8:end8:stride8]

instead of ARRAY[[s1:e1:str1],[s2:e2:str2], ...]
So, I don't think A[:3:] would (and should) be different from A[:[3]:]
You'll probably have to stick with good old REFORM for this.

> 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

More generally, this points to the problem of converting 1-dimensional index arrays (as from WHERE) to multi-dimensional arrays and vice versa.
We had a related discussion in this group a while ago. If I remember correctly, this was about what people expect from
A[ind1, ind2, ind3] where ind1, ind2, ind3 are 1-dimensional vectors > 1 element.

Here is what I see:
(1) multi-dimensional index

```
a = findgen(10,10,10)
b = lonarr(2,3,4)
; fill b with some values
b[*,1,4] = 3
help,a[b]
print,a[b]
```

BUT is b not in fact interpreted as a 1-D index? Suspicion arises because a[b,1,1] will
also work (and return a 1D array).

(2) combi of 1-dimensional indices

```
a = findgen(10,10,10)
b1 = [1,2]
b2 = [2,3]
b3 = [4,8]   ; don't try b3=[3,6,7] !
help,a[b]
print,a[b]
```

So, YES! It would be nice if one could use a multi-dimensional array index, but there are several pitfalls here, and it appears as a non-trivial problem.

Regards,
Martin

--

||||||||||||||\\\\\\\\\\\\\\----------////////////// //||||||||||||||
Martin Schultz, DEAS, Harvard University, 29 Oxford St., Pierce 109,
Cambridge, MA 02138 phone (617) 496 8318 fax (617) 495 4551
e-mail mgs@io.harvard.edu web http://www-as/people/staff/mgs/

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

Stein Vidar Hagfors Haugan wrote:
> 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?

I agree that START, STRIDE, COUNT are somewhat wordy. However I'd like to be able to specify them in any combination, e.g.
START only, or
STRIDE only, or
COUNT only, or
START and STRIDE, or
START and COUNT, or
STRIDE and COUNT, or
START and STRIDE and COUNT.
I'm not sure I know a clean way to allow these combinations, other than using optional keywords. Does anyone else have any thoughts? If optional positional parameters were used, there would have to be a hierarchy, as Stein suggests below.

> 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.

How about ARRGET and ARRPUT? (Sounds a bit like FORTRAN-77 to me...)

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

If it was done this way, I'd prefer that the positional parameters be defined the same way as in my original spec.

> 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.

If you use the keyword method, then omitting the keyword means read 'everything' in the START, STRIDE, or COUNT context.

> 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)

This could always be an option. The default method could be to leave dangling dimensions, with an optional keyword Boolean flag that specifies all dangling dimensions be removed. I think leaving the dangling dimensions alone by default is a good idea, since you might want to modify the extracted array, and then re-insert it back into the original array. That might be difficult if the dangling dimensions are lost.

> 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

That shouldn't be a problem.

Liam E. Gumley
Space Science and Engineering Center, UW-Madison
http://cimss.ssec.wisc.edu/~gumley

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

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

Liam Gumley, Liam.Gumley@ssec.wisc.edu writes:

> Comments are invited.

I think it's a great idea, particularly if RSI can be persuaded to write it as an efficient internal function. I agree with Martin that it might be better to call it ARRAY_EXTRACT. It would also be nice to have a matching ARRAY_INSERT procedure that accepts the same START, STRIDE and COUNT descriptors.

Struan

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

Liam Gumley wrote:
>
> 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.
>
> Cheers,
> Liam.
>
> [header snipped]

Nice plan, Liam!

Only that it interferes what I would call "slicing", i.e. extraction of

an N-1 dimensional hypersurface from an N dimensional array. Why not "array_extract"?

Regards,
Martin.

PS: thanks for your mails.

--

|||||||||||||||\\\\\\\\\\\\\\\----------////////////// //|||||||||||||||
Martin Schultz, DEAS, Harvard University, 29 Oxford St., Pierce 109,
Cambridge, MA 02138 phone (617) 496 8318 fax (617) 495 4551
e-mail mgs@io.harvard.edu web http://www-as/people/staff/mgs/