Subject: What are the rules for automatic removal of singleton dimensions, and can I have a way of disabling them, please?
Posted by tom.grydeland on Sun, 23 Dec 2012 23:01:24 GMT
View Forum Message <> Reply to Message

Hello all,

I was trying to visualize subsections of a windowing function when this bit me.

I was creating piecewise results in an array res[ix, iy, ii, jj], where the partial results were sums up to some value in the final two indices

```
for ii = 0, Kx-1 do begin
  for jj = 0, Ky-1 do begin
    visualize, total(total(res[*,*,0:ii,0:jj], 4), 3)
  endfor
endfor
```

but the problem is that IDL (arbitrarily, IMO) discards trailing singleton dimensions on my indexing, so that res[*,*,0:ii,0:jj] ends up as a two-dimensional array when both ii and jj are zero, and a three-dimensional index on subsequent cases of jj being zero, which again causes the calls to 'total' to fail. The innermost portion of these loops become extraordinarily messy if trying to fix this problem.

I've fixed my program by reordering the indices (to [ii, jj, ix, iy]), but I am still miffed that this should be necessary.

Similarly, when I concatenate arrays of dimensions [x, j] and [y, j], I expect a result with dimensions [x+y, j], even when j is equal to 1. I'm trying to write programs independent of the actual value of j, but these arbitrary removals of singleton dimensions make my task that much harder.

Is there a way to disable this stripping of singleton dimensions?

--Tom Grydeland

Subject: Re: What are the rules for automatic removal of singleton dimensions, and can I have a way of disabling them, please?
Posted by David Fanning on Wed, 26 Dec 2012 01:47:44 GMT
View Forum Message <> Reply to Message

Tom Grydeland writes:

> For sure I meant a compile_opt, I understand that default behavior cannot be changed for something like this.

It's not even clean to me that a complier option can

fix what is surely a run-time operation.

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
Sepore ma de ni thue. ("Perhaps thou speakest truth.")

---

Subject: Re: What are the rules for automatic removal of singleton dimensions, and can I have a way of disabling them, please?
Posted by on Wed, 26 Dec 2012 09:25:09 GMT
View Forum Message <> Reply to Message

Den måndagen den 24:e december 2012 kl. 00:01:24 UTC+1 skrev Tom Grydeland:
> Hello all,
>
>
>
> I was trying to visualize subsections of a windowing function when this bit me.
>
>
>
> I was creating piecewise results in an array res[ix, iy, ii, jj], where the partial results were sums up to some value in the final two indices
>
>
>
> for ii = 0, Kx-1 do begin
>
> for jj = 0, Ky-1 do begin
>
> visualize, total(total(res[*,*,0:ii,0:jj], 4), 3)
>
> endfor
>
> endfor
>
>
>
> but the problem is that IDL (arbitrarily, IMO) discards trailing singleton dimensions on my indexing, so that res[*,*,0:ii,0:jj] ends up as a two-dimensional array when both ii and jj are zero,

and a three-dimensional index on subsequent cases of jj being zero, which again causes the calls to 'total' to fail.  The innermost portion of these loops become extraordinarily messy if trying to fix this problem.
>
>
>
>  I've fixed my program by reordering the indices (to [ii, jj, ix, iy]), but I am still miffed that this should be necessary.
>
>
>
>  Similarly, when I concatenate arrays of dimensions [x, j] and [y, j], I expect a result with dimensions [x+y, j], even when j is equal to 1.  I'm trying to write programs independent of the actual value of j, but these arbitrary removals of singleton dimensions make my task that much harder.
>
>
>
>  Is there a way to disable this stripping of singleton dimensions?

I thought the idea with the stripping of dimensions was that it shouldn't matter whether these dimensions are there or not. I mena, you can still address a[2,2,0] when a=fltarr(5,5). So shouldn't the fix really be to make total() work for non-existing trailing dimensions?

(Or, if just to make your code less messy, to write a mytotal() function that takes care of the exceptions.)

---

Subject: Re: What are the rules for automatic removal of singleton dimensions, and can I have a way of disabling them, please?
Posted by tom.grydeland on Wed, 26 Dec 2012 14:12:35 GMT
View Forum Message <> Reply to Message

On Wednesday, December 26, 2012 10:25:09 AM UTC+1, Mats Löfdahl wrote:

>  I thought the idea with the stripping of dimensions was that it shouldn't matter whether these dimensions are there or not. I mena, you can still address a[2,2,0] when a=fltarr(5,5). So shouldn't the fix really be to make total() work for non-existing trailing dimensions?

In my mind, the fix is to make things behave _less_ arbitrarily, not _more_ so.  I would happily accept indexing that raised exceptions on nonexistent dimensions, if I could rely on the dimensions not disappearing when I didn't ask for it.

A related and tangential question for the IDL internals wizards:  Can the evaluator distinguish between single-valued indexes and a single-element range index?  I.e. can it tell the difference between a[*,0] and a[*,0:ii] when ii eq 0?

>  (Or, if just to make your code less messy, to write a mytotal() function that takes care of the

exceptions.)

That solution is worse than the problem, IMO.  How many other functions will I have to replace?

--T

---

Subject: Re: What are the rules for automatic removal of singleton dimensions, and can I have a way of disabling them, please?
Posted by David Fanning on Wed, 26 Dec 2012 16:36:51 GMT
View Forum Message <> Reply to Message

Tom Grydeland writes:

> A related and tangential question for the IDL internals wizards:  Can the evaluator distinguish between single-valued indexes and a single-element range index?  I.e. can it tell the difference between a[*,0] and a[*,0:ii] when ii eq 0?

Doesn't it take longer to ask the question than it does to
do the experiment?

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
Sepore ma de ni thue. ("Perhaps thou speakest truth.")

---

Subject: Re: What are the rules for automatic removal of singleton dimensions, and can I have a way of disabling them, please?
Posted by  on Wed, 26 Dec 2012 23:12:40 GMT
View Forum Message <> Reply to Message

Den onsdagen den 26:e december 2012 kl. 15:12:35 UTC+1 skrev Tom Grydeland:
> On Wednesday, December 26, 2012 10:25:09 AM UTC+1, Mats Löfdahl wrote:
>
>> I thought the idea with the stripping of dimensions was that it shouldn't matter whether these dimensions are there or not. I mena, you can still address a[2,2,0] when a=fltarr(5,5). So shouldn't the fix really be to make total() work for non-existing trailing dimensions?
>
> In my mind, the fix is to make things behave _less_ arbitrarily, not _more_ so.

So then we agree.

>> (Or, if just to make your code less messy, to write a mytotal() function that takes care of the exceptions.)
>
> That solution is worse than the problem, IMO. How many other functions will I have to replace?

How would I know? Total() is the only one you mentioned...

---

Subject: Re: What are the rules for automatic removal of singleton dimensions, and can I have a way of disabling them, please?
Posted by tom.grydeland on Thu, 27 Dec 2012 23:06:56 GMT
View Forum Message <> Reply to Message

On Thursday, December 27, 2012 12:12:40 AM UTC+1, Mats Löfdahl wrote:
> Den onsdagen den 26:e december 2012 kl. 15:12:35 UTC+1 skrev Tom Grydeland:

>> In my mind, the fix is to make things behave _less_ arbitrarily, not _more_ so.
> So then we agree.

Good!

>> That solution is worse than the problem, IMO. How many other functions will I have to replace?
> How would I know? Total() is the only one you mentioned...

Yes -- I believe in making my posted examples as succinct as possible.

You pointed out something of which I wasn't aware -- that adding an extra [ ... , 0] index does not raise an exception or change the returned values. This is valuable information, but not something I would base my code on. In my mind, it means indexing has one special case with erratic behavior. Before I start modifying (parts of) the IDL standard library to also behave erratically (and then have to keep track of which parts are so modified), I would try almost any other solution which mean that my array has the number of dimensions I want before I start:

```
pro atleast_n_dim, a, n
   adim = size(a, /dimensions)
   if n_elements(adim) eq n then return

   adim = [adim, 1+intarr(n-n_elements(adim))]
   a = reform(a, adim, /overwrite)
   return
end
```

--T

---

Subject: Re: What are the rules for automatic removal of singleton dimensions, and can I have a way of disabling them, please?
Posted by on Fri, 28 Dec 2012 10:28:09 GMT
View Forum Message <> Reply to Message

Den fredagen den 28:e december 2012 kl. 00:06:56 UTC+1 skrev Tom Grydeland:
> On Thursday, December 27, 2012 12:12:40 AM UTC+1, Mats Löfdahl wrote:
>
>> Den onsdagen den 26:e december 2012 kl. 15:12:35 UTC+1 skrev Tom Grydeland:
>
>>> In my mind, the fix is to make things behave _less_ arbitrarily, not _more_ so.
>
>> So then we agree.
>
> Good!

I guess we just do not agree on what is less or more arbitrary... :o)

I agree that it may have been a bad design decision to make IDL arrays behave in this way in the first place, although I don't know all the reasons behind it. But, as has been pointed out, the chance that IDL's default behavior would be changed in this respect is slim. What is then less arbitrary, to introduce a compile option that would make your code more difficult to read (because it would work differently from other IDL code) or to make core IDL functions work consistently with the way arrays are designed? Apparently we answer this question differently and that is fine.

So, while I don't begrudge you the solution you like, if that compile option is implemented I probably would not use it. And if it is implemented, I think it would be a good idea to make total() and other similar functions work properly anyway. Median() with the dimension keyword for example. I have sometimes wondered why mean and stddev doesn't have that keyword, but now I see that it was introduced in IDL v 8. I don't have that version so I can't test how extra dimensions are handled for those functions but I guess they have the same problem as total() and median().

For the functions I mentioned above, the exception would be very simple to handle. Unless my thinking is completely off today, total(), mean(), and median() should just return the input array if the requested dimension is larger than the number of dimensions, while stddev() should return an array of the same type and dimensions but filled with zeros. Shouldn't be too hard do figure out how to handle the higher moments.

Or would this lead to problems in other parts of IDL?

>>> That solution is worse than the problem, IMO. How many other functions will I have to replace?
>
>> How would I know? Total() is the only one you mentioned...
>
> Yes -- I believe in making my posted examples as succinct as possible.

Fine, I just didn't realize it was only an example. I agree that the problem should not be solved for total() only. I just meant that if the particular code you are working on for the moment suffers from

this problem only with the total() function, your code might be easier to read (and possibly faster if you now feel you have to reorder the dimensions for large arrays?) by making a private version of total() with the simple exception I mention above.

> You pointed out something of which I wasn't aware -- that adding an extra [ ... , 0] index does not raise an exception or change the returned values. This is valuable information, but not something I would base my code on.

I think you could. It seems as unlikely that this would change as it is that the automatically removed dimensions would be ... well ... removed.

---

Subject: Re: What are the rules for automatic removal of singleton dimensions, and can I have a way of disabling them, please?
Posted by tom.grydeland on Wed, 16 Jan 2013 11:09:45 GMT
View Forum Message <> Reply to Message

Hi again,

On Monday, December 24, 2012 12:01:24 AM UTC+1, Tom Grydeland wrote:
> I was trying to visualize subsections of a windowing function when this bit me.

Today, it bit me again.

I was fiddling with a small routine to create dense grid index arrays (akin to those the 'mgrid' object from NumPy create), using a combination of adding extra indices and transpose() to create my matrices. If I could trust IDL not to strip dimensions, I could write this cleanly as:

(ax, ay and az are all vectors of length >= 1 at this point)

```
xout = ax[*,ay,az]
yout = transpose(ay[*,ax,az], [1,0,2])
zout = transpose(az[*,ax,ay], [1,2,0])
```

Unfortunately, the gratuitous stripping of dimensions even inside the indexing expression means that the array given to transpose() doesn't have three dimensions anymore, and I have to resort to this mess, which is much less readable and much harder to maintain.

```
out_dims = [n_elements(ax), n_elements(ay), n_elements(az)]
xout = reform(ax[*,ay,az], out_dims)
yout = transpose(reform(ay[*,ax,az], out_dims[[1, 0, 2]]), [1, 0, 2])
zout = transpose(reform(az[*,ax,ay], out_dims[[2, 0, 1]]), [1, 2, 0])
```

(Of course, in line with Mats Löfdahl's suggestion earlier, TRANSPOSE could also be modified to add back singleton dimensions to its input array as necessary.)

--Tom Grydeland

Subject: Re: What are the rules for automatic removal of singleton dimensions, and can I have a way of disabling them, please?
Posted by tom.grydeland on Thu, 17 Jan 2013 08:11:40 GMT
View Forum Message <> Reply to Message

On Wednesday, January 16, 2013 12:09:45 PM UTC+1, Tom Grydeland wrote:

Since I'm talking to myself here, I might as well reply to myself also.

> (Of course, in line with Mats Löfdahl's suggestion earlier, TRANSPOSE could also be modified to add back singleton dimensions to its input array as necessary.)

Having written this, I decided to go ahead and do it that way also, just to see. It feels a little bit like I'm fixing IDL's bugs for them instead of working on the problems I'm actually paid to solve. I tried writing MY_TRANSPOSE so that it can be a drop-in replacement of TRANSPOSE, i.e. it shall give exactly the same results as TRANSPOSE for every input that TRANSPOSE will accept.

The tedium of adding singleton dimensions was factored out to a function ATLEAST_N_DIM, which modifies and returns its first argument.

I'm putting these functions out there for anyone to see or use as they see fit.

I am still hopeful that I can get a COMPILE_OPT that would (within a certain scope) disable stripping of trailing singleton dimensions, so I wouldn't have to resort to such hackery or the replacement of base IDL functionality.

Stylistic comments, suggestions, etc?

```
;+
; Small utility that ensures its input array has dimensionality at least N.
; Modifies _and_ returns its input, without making copies.
;
; :Params:
;   a: in, out
;     array to modify dimensionality of
;   n: in, type=integer
;     minimum number of dimensions for result
;
; :Returns:
;   a, with zero or more singleton dimensions appended
;
; :See also:
;   MY_TRANSPOSE, DENSEGRID
;-
```

```
function atleast_n_dim, a, n
   adim = size(a, /dimensions)
   if n_elements(adim) ge n then return, a

   adim = [adim, make_array(n-n_elements(adim), value=1)]
   a = reform(a, adim, /overwrite)
   return, a
end

;+
; Version of TRANSPOSE that will add singleton dimensions if needed
;
; :See also:
;   ATLEAST_N_DIM, DENSEGRID
;-
function my_transpose, arr, perm
   ;; defer simple cases to native TRANSPOSE
   if n_elements(perm) eq 0 then return, transpose(arr)

   adims = size(arr, /dimensions)
   np = max(perm)+1
   na = n_elements(adims)
   if np le na then return, transpose(arr, perm)

   narr = transpose(atleast_n_dim(arr, np), perm)
   ;; set dimensions of arr back to what they were
   arr = reform(arr, adims, /overwrite)
   return, narr
end
```

---

Subject: Re: What are the rules for automatic removal of singleton dimensions, and can I have a way of disabling them, please?
Posted by Jeremy Bailin on Thu, 17 Jan 2013 14:58:34 GMT
View Forum Message <> Reply to Message

On 1/17/13 2:11 AM, Tom Grydeland wrote:
> On Wednesday, January 16, 2013 12:09:45 PM UTC+1, Tom Grydeland wrote:
>
> Since I'm talking to myself here, I might as well reply to myself also.
>
>> (Of course, in line with Mats Lï¿½fdahl's suggestion earlier, TRANSPOSE could also be modified to add back singleton dimensions to its input array as necessary.)
>
> Having written this, I decided to go ahead and do it that way also, just to see. It feels a little bit like I'm fixing IDL's bugs for them instead of working on the problems I'm actually paid to solve. I tried writing MY_TRANSPOSE so that it can be a drop-in replacement of TRANSPOSE, i.e. it shall give exactly the same results as TRANSPOSE for every input that TRANSPOSE will accept.

>
> The tedium of adding singleton dimensions was factored out to a function ATLEAST_N_DIM, which modifies and returns its first argument.
>
> I'm putting these functions out there for anyone to see or use as they see fit.
>
> I am still hopeful that I can get a COMPILE_OPT that would (within a certain scope) disable stripping of trailing singleton dimensions, so I wouldn't have to resort to such hackery or the replacement of base IDL functionality.
>
> Stylistic comments, suggestions, etc?
>
>
> ;+
> ; Small utility that ensures its input array has dimensionality at least N.
> ; Modifies _and_ returns its input, without making copies.
> ;
> ; :Params:
> ;   a: in, out
> ;     array to modify dimensionality of
> ;   n: in, type=integer
> ;     minimum number of dimensions for result
> ;
> ; :Returns:
> ;   a, with zero or more singleton dimensions appended
> ;
> ; :See also:
> ;   MY_TRANSPOSE, DENSEGRID
> ;-
> function atleast_n_dim, a, n
>      adim = size(a, /dimensions)
>      if n_elements(adim) ge n then return, a
>
>      adim = [adim, make_array(n-n_elements(adim), value=1)]
>      a = reform(a, adim, /overwrite)
>      return, a
> end
>
> ;+
> ; Version of TRANSPOSE that will add singleton dimensions if needed
> ;
> ; :See also:
> ;   ATLEAST_N_DIM, DENSEGRID
> ;-
> function my_transpose, arr, perm
>      ;; defer simple cases to native TRANSPOSE
>      if n_elements(perm) eq 0 then return, transpose(arr)
>

```
>       adims = size(arr, /dimensions)
>       np = max(perm)+1
>       na = n_elements(adims)
>       if np le na then return, transpose(arr, perm)
>
>       narr = transpose(atleast_n_dim(arr, np), perm)
>       ;; set dimensions of arr back to what they were
>       arr = reform(arr, adims, /overwrite)
>       return, narr
>  end
>
```

Looks good to me - I will definitely use them (at least the utility function).

-Jeremy.

---

Subject: Re: What are the rules for automatic removal of singleton dimensions, and can I have a way of disabling them, please?
Posted by David Fanning on Thu, 17 Jan 2013 15:17:20 GMT
View Forum Message <> Reply to Message

Tom Grydeland writes:

> It feels a little bit like I'm fixing IDL's bugs for them instead of working on the problems I'm actually paid to solve.

It is possible to make a career of this. You should consider it. ;-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
Sepore ma de ni thue. ("Perhaps thou speakest truth.")

---