Subject: Array multiplication: implicit loop query
Posted by george on Fri, 10 Aug 2001 10:56:55 GMT
View Forum Message <> Reply to Message

Hi there

I was just calculating the following equation:

DEN_H = MMR_H * Pres * RMT / (atomic_mass_H * Gas_constant * TN)

These numbers are 3D arrays, 1D arrays and constants, i.e.,

MMR_H = fltarr(30,91,40)
Pres = fltarr(30)
RMT = fltarr(30,91,40)
atomic_mass_H = constant
Gas_constant = constant
TN =fltarr(30,91,40)

The result of this is DEN_H (previously undefined) which ends up being
fltarr(30) - i.e., 1 dimensional.
To my mind DEN_H should be 3D (30,91,40) - shouldn't it ? Doesn't IDL
understand that I am implicity doing a full 3D calculation here ? It
would seem that, to get this to work I need to make
Pres=fltarr(30,91,40). However, this seems uneccessary because that
means that the 2 new dimensions for pressure are redundant.
Any advice ?? Cheers,
George Millward

Subject: Re: Array multiplication: implicit loop query
Posted by Paul van Delst on Mon, 13 Aug 2001 13:46:06 GMT
View Forum Message <> Reply to Message

george Millward wrote:
>
> Hi there,
>
> Thanks for everyones help.
>
> I have inserted the "rebin" function and this works fine.
> I am still a little intrigued as to why IDL works this way

I'm sorta intrigued why you think it would work the other way. Somewhere someone had to
make a decision what to do in this respect. What would expect the default behaviour to be
if you mixed not just 1-D and 3-D arrays, but had a couple of 2-d or 4-d arrays also? Or
if you had some subscripted arrays in the expression? Which dimensions get "promoted" or
"rebinned" with respect to others? Seems to me like using the lowest common dimensions

makes more sense - not just for implementation but also for spotting errors when you've mixed up your dimensions. It, uhh, ehem, happens to me all the time... :oD

paulv

--
Paul van Delst A little learning is a dangerous thing;
CIMSS @ NOAA/NCEP Drink deep, or taste not the Pierian spring;
Ph: (301)763-8000 x7274 There shallow draughts intoxicate the brain,
Fax:(301)763-8545 And drinking largely sobers us again.
Alexander Pope.

Subject: Re: Array multiplication: implicit loop query
Posted by Craig Markwardt on Mon, 13 Aug 2001 13:58:08 GMT
View Forum Message <> Reply to Message

george@apg.ph.ucl.ac.uk (george Millward) writes:
>
> I have inserted the "rebin" function and this works fine.
> I am still a little intrigued as to why IDL works this way - it still
> seems to me that my orginal combination of 3D and 1D arrays should
> yield a 3D array. Not a problem - we all live with "features" of
> programming languages - just wondering.

Heh, the reason is pretty simple, if not intuitive. When confronted with operations between arrays of different sizes, IDL will *truncate* the longest array to the shortest size. I think this rule is pretty general but somebody will probably pipe in with an exception.

That "limitation" can sometimes be used to your advantage. For example, when doing finite differencing, a[i+1] - a[i] for each element, normally you would write that like:

```
diff = a(1:*) - a(0:n_elements(a)-2)
```

The expression A(0:N_ELEMENTS(A)-2) removes that last element of the array A. But that is a little obscure. It's almost "better" to say,

```
diff = a(1:*) - a(0:*)  ; or
diff = a(1:*) - a
```

It saves keystrokes, avoids subscript clutter, etc. But why does it work? The answer is that A(1:*) has one less element than A, so when A appears by itself in the above equation, it is automatically truncated by one, achieve the desired result.

Craig

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

Subject: Re: Array multiplication: implicit loop query
Posted by david[2] on Mon, 13 Aug 2001 14:55:04 GMT
View Forum Message <> Reply to Message

JD Smith writes:
>
> function where_not_array,A,B,cnt,IA_IN_B=iA_in_B
>
> Na = n_elements(a)
> Nb = n_elements(b)
> l = lindgen(Na,Nb)
> AA = A(l mod Na)
> BB = B(l / Na)
>
> if keyword_set(iA_in_B) then $
> wh = where(total(AA ne BB,2) eq Nb,cnt) $
> else wh = where(total(AA ne BB,1) eq Na,cnt)
>
> return,wh
> end

It boggles the mind, that's for sure. Even
without Histograms. :-(

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: Array multiplication: implicit loop query
Posted by Richard Younger on Mon, 13 Aug 2001 18:37:11 GMT
View Forum Message <> Reply to Message

JD Smith wrote:
>

[...]

>
> Don't abandon those subscripting array inflation techniques just yet
> though! While rebin/reform is conceptually simpler (especially for more
> than 2 dimensions), the old lindgen() method still has its place. When,
> you ask? Well, rebin works only with numeric data. If you have an
> array of structures, pointers, or objects, you'll need to fall back on
> the ancestral methods.
>
> The idea is simple. Construct an array of indices of the size you're
> after, and use "mod" and "/" to massage it into the correct form for
> indexing into the original array. If you have many such arrays to
> inflate, it may even be competitive in speed (since you have to
> precompute the index array only once).
>
> In 2D it's simple.
>
> IDL> a=findgen(5)
> IDL> inds=lindgen(5,10)
> IDL> big_a=a[inds mod 5] ; across
> IDL> inds=lindgen(10,5)
> IDL> big_a=a[inds/10] ; down
>
> for higher dimensions, it quickly becomes cumbersome (try it and see).
>
> JD
[...]

Wow. Thanks, JD. This looks like a wonderfully useful technique. Consequently, I want to try to figure out for myself how it works in a more general way. :-) It took me a little while working out examples to get a start, but let me see if my mental model is at all accurate.

It seems this method in multiple dimensions requires indexing using only one index dimension (i.e. A[6] instead of A[0,1,2]). This makes things complicated, since modulo works nicely for cutting out the last dimension, and division works nicely for cutting out the first, but for getting stuff in between you have to do a combination of both.

conventions for the next few statements:
-take an array A with dimensions 1,2,...,n and dimension sizes S[1,2,...,n]
-you want to insert an expanded dimension of size Se

Therefore, if you want to put the existing array in dimensions (2 ->
n+1) and expand on dimension 1, you can create an index array with
dimensions (Se, S[1], S[2],...,S[n]), and divide your index array by Se.

If you want to put the existing array in dimensions (1 - > n) and expand
on dimension n+1, you can create an index array with dimensions (S[1],
S[2],...,S[n], Se) and take the modulo with Product(S[1],...,S[n]).

And your final array is just A with indices as above. I think you could
expand multiple dimensions at the beginning or end by replacing Se with
[Se1, Se2, ...] and the division with Se by Prod(Se1, Se2, ...), but I
haven't tried it out.

However, if you want to expand on some interior dimension, call it k,
things get complicated. It seems to me that you'd have to insert the
new dimension at k in the index array and do some combination of
division and modulo arithmetic (perhaps modulo Prod(S[1],..., S[k]) and
division by (?) -- it would have to reduce to the two cases above) to
get things to work out. Can't you just do one of the above and
TRANSPOSE() to get the desired array? Hmm. I'm sure someone's figured
this out already.

Thoughts?

Rich

P.S. On a side note, I recently reexamined histogram and I think I'm
beginning to understand it. The breakthrough from total opacity came
when I stopped thinking of REVERSE_INDICES as an array of numbers and
started thinking of it as a data structure of pointers. Now I can start
puzzling through those histogram examples instead of just having my eyes
glaze over at the magic of it all.

--
Richard Younger

Subject: Re: Array multiplication: implicit loop query
Posted by david[2] on Mon, 13 Aug 2001 18:52:17 GMT
View Forum Message <> Reply to Message

Richard Younger writes:

> P.S. On a side note, I recently reexamined histogram and I think I'm
> beginning to understand it. The breakthrough from total opacity came
> when I stopped thinking of REVERSE_INDICES as an array of numbers and
> started thinking of it as a data structure of pointers. Now I can start

> puzzling through those histogram examples instead of just having my eyes
> glaze over at the magic of it all.

This was my problem, too!

I was thinking of the numbers as unforced
forehand errors in my last tennis match
and the enormity of the problem soon
overwhelmed me. Now that I'm thinking
correctly, the vale has lifted. :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: Array multiplication: implicit loop query
Posted by Richard Younger on Mon, 13 Aug 2001 20:59:59 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>
> Richard Younger writes:
>
>> P.S. On a side note, I recently reexamined histogram and I think I'm
>> beginning to understand it. The breakthrough from total opacity came
>> when I stopped thinking of REVERSE_INDICES as an array of numbers and
>> started thinking of it as a data structure of pointers. Now I can start
>> puzzling through those histogram examples instead of just having my eyes
>> glaze over at the magic of it all.
>
> This was my problem, too!
>
> I was thinking of the numbers as unforced
> forehand errors in my last tennis match
> and the enormity of the problem soon
> overwhelmed me. Now that I'm thinking
> correctly, the vale has lifted. :-)

David,

Well, you win some, you loose some. No matter how hard I concentrate on turning my list titled 'Features to add to application' into 'Trivial tasks done by my computer while I sleep', it never seems to shrink any faster. :-(

Anyone up for "incomprehensible discussions on Histogram functionality?" I can be incomprehensible, but I'm a little short (okay, WAY short) on the Histogram functionality.

Rich

--
Richard Younger

Subject: Re: Array multiplication: implicit loop query
Posted by bennetsc on Wed, 15 Aug 2001 06:57:32 GMT
View Forum Message <> Reply to Message

In article <onelqg12q7.fsf@cow.physics.wisc.edu>,
Craig Markwardt <craigmnet@cow.physics.wisc.edu> wrote:
>
> george@apg.ph.ucl.ac.uk (george Millward) writes:
>>
>> I have inserted the "rebin" function and this works fine.
>> I am still a little intrigued as to why IDL works this way - it still
>> seems to me that my orginal combination of 3D and 1D arrays should
>> yield a 3D array. Not a problem - we all live with "features" of
>> programming languages - just wondering.
>
> Heh, the reason is pretty simple, if not intuitive. When confronted
> with operations between arrays of different sizes, IDL will *truncate*
> the longest array to the shortest size. I think this rule is pretty
> general but somebody will probably pipe in with an exception.

    Yup. See below.
>
> That "limitation" can sometimes be used to your advantage. For
> example, when doing finite differencing, a[i+1] - a[i] for each
> element, normally you would write that like:
>
> diff = a(1:*) - a(0:n_elements(a)-2)
>
> The expression A(0:N_ELEMENTS(A)-2) removes that last element of the
> array A. But that is a little obscure. It's almost "better" to say,
>
> diff = a(1:*) - a(0:*) ; or

> diff = a(1:*) - a

Here's an exception, sort of:

 a[*] = a[1:*] - a

The above will, in fact, get you an error message from IDL to the effect that n_elements(RHS) ne n_elements(LHS). For consistency with Craig's examples, one might like to see IDL understand that this example should be handled in the same manner as

 a[0:n_elements(a[1:*] - a) - 1] = a[1:*] - a
             ^^^^^^^^^^^^   ^^^^^^^^^^^^

In other words, one might want IDL to truncate the vector length meant by "*" on the LHS to match the shorter RHS vector length.
>
> It saves keystrokes, avoids subscript clutter, etc. But why does it

That it does. Craig's latter example--the one with "- a" instead of "- a[0:n_elements(a) - 2]" may also get you some nasty surprises, perhaps undetected till too late, if you ever change earlier parts of the program in certain ways.

> work? The answer is that A(1:*) has one less element than A, so when
> A appears by itself in the above equation, it is automatically
> truncated by one, achieve the desired result.
>
That's right, but incomplete. The exception, as noted above, is that the truncation of vector length in an assignment statement applies *only* to the RHS. One might argue that the exception does not apply in a case like

> diff = a(1:*) - a(0:*) ; or
> diff = a(1:*) - a

but this is because a new target is being created to which the evaluated expression will be assigned, rather than storing the evaluated expression into part of an existing target. n_elements(diff) is undefined until after the first time the assignment is completed, so it's really not the same situation.

Scott Bennett, Comm. ASMELG, CFIAG
College of Oceanic and Atmospheric
Sciences
Oregon State University
Corvallis, Oregon 97331

```
 ************************************************************ **********
* Internet:       sbennett at oce.orst.edu                          *
 *------------------------------------------------------------ ---------*
* "Lay then the axe to the root, and teach governments humanity.     *
* It is their sanguinary punishments which corrupt mankind."         *
*    -- _The_Rights_of_Man_ by Tom Paine (1791.)                     *
 ************************************************************ **********
```

Subject: Re: Array Multiplication
Posted by lecacheux.alain on Sun, 25 Mar 2012 15:09:57 GMT
View Forum Message <> Reply to Message

On 25 mar, 11:22, IDL beginner <moxam...@gmail.com> wrote:
> Dear All,
>
> I have the following question. Let's say we have a vector A and matrix
> array B so that:
>
> A = [1, 2, 5, 2, 3]
>
> B = [[2, 4], [1, 9]]
>
> I want to multiply each element of the vector A with the matrix array
> B but WITHOUT using a FOR loop. Is that possible??? I need to do so in
> order to save execution time.
>
> The output result should be an array C with dimensions C[2, 2, 5, 1].
>
> Any help is appreciated.
>
> MD
>
>

C = reform(reform(B, 4, OVER=over) # A, 2, 2, 5)

Regarding any optimization, using over=1 is faster, but will change
dimensions of B; using over=0 will keep B unchanged, but a temporary
copy of B is created.
alx.

Subject: Re: Array Multiplication
Posted by Craig Markwardt on Sun, 25 Mar 2012 19:58:53 GMT
View Forum Message <> Reply to Message

On Sunday, March 25, 2012 5:22:00 AM UTC-4, IDL beginner wrote:
> Dear All,
>
> I have the following question. Let's say we have a vector A and matrix
> array B so that:
>
> A = [1, 2, 5, 2, 3]
>
> B = [[2, 4], [1, 9]]
>
> I want to multiply each element of the vector A with the matrix array
> B but WITHOUT using a FOR loop. Is that possible??? I need to do so in
> order to save execution time.
>
> The output result should be an array C with dimensions C[2, 2, 5, 1].

C = intarr(2,2,5,1)
C[0,0,*,*] = B[0,0]*A
C[0,1,*,*] = B[0,1]*A
C[1,0,*,*] = B[1,0]*A
C[1,1,*,*] = B[1,1]*A

Since there are only four elements, there are no FOR loops needed.

I suppose that you are trying to frame a more generic problem. My recommendation is at first, don't worry about whether you are using FOR loops or not. Make each iteration of the loop do as much work as possible, and you should be fine.

Craig