Subject: Re: Array multiplication: implicit loop query
Posted by billb on Fri, 10 Aug 2001 12:53:25 GMT
View Forum Message <> Reply to Message

george@apg.ph.ucl.ac.uk (george Millward) wrote in message
news:<d90c0773.0108100256.6398a693@posting.google.com>...
> 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 ?

No.

> It
> would seem that, to get this to work I need to make
> Pres=fltarr(30,91,40).

Yes.

```
IDL> a = indgen(20,20)
IDL> b = indgen(20)
IDL> c = b * a
IDL> help, c
C               INT       = Array[20]
```

I believe you need to REPLICATE 'Pres' as needed.

-Bill B.

Subject: Re: Array multiplication: implicit loop query

Posted by Kenneth P. Bowman on Fri, 10 Aug 2001 12:58:03 GMT
View Forum Message <> Reply to Message

In article <d90c0773.0108100256.6398a693@posting.google.com>, george Millward <george@apg.ph.ucl.ac.uk> wrote:

```
>  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 ?
```

No.

```
>  It would seem that, to get this to work I need to make
>  Pres=fltarr(30,91,40).
```

That's one solution. The other is to use loops.

```
FOR k = 0, 29 DO DEN_H[k,*,*] = MMR_H[k,*,*]*Pres[k]*RMT[k,*,*] / $
                 (atomic_mass_H* Gas_constant*TN[k,*,*])
```

This would be *much* more efficient (in terms of cache usage and array indexing) if pressure was your last dimension, i.e.,

```
MMR_H = fltarr(91,40,30)
Pres = fltarr(30)
RMT = fltarr(91,40,30)
TN =fltarr(91,40,30)
```

Then you can write

```
FOR k = 0, 29 DO DEN_H[0,0,k] = MMR_H[*,*,k]*Pres[k]*RMT[*,*,k] / $
                 (atomic_mass_H*Gas_constant*TN[*,*,k])
```

The change in the indexing on the LHS to [0,0,k] is important.

Ken

Subject: Re: Array multiplication: implicit loop query
Posted by Richard Younger on Fri, 10 Aug 2001 15:09:46 GMT
View Forum Message <> Reply to Message

"Bill B." wrote:
>
> george@apg.ph.ucl.ac.uk (george Millward) wrote in message

```
>> It
>> would seem that, to get this to work I need to make
>> Pres=fltarr(30,91,40).
>
> Yes.
>
> IDL> a = indgen(20,20)
> IDL> b = indgen(20)
> IDL> c = b * a
> IDL> help, c
> C               INT       = Array[20]
>
> I believe you need to REPLICATE 'Pres' as needed.
>
> -Bill B.
```

I've been converted to REBIN, myself.
(see the group archives for the dimensional juggling tutorial by JD
Smith this past spring)

```
IDL> Pres_expan = REBIN(Pres, 30, 91, 40)
IDL> help, Pres_expan
PRES_EXPAN      FLOAT     = Array[30, 91, 40]
```

This gives you the correct dimensions and avoids loops. You can even
resize inline if you dislike having the extra variable around:

```
DEN_H = MMR_H * REBIN(Pres, 30, 91, 40) * RMT / $
 ( atomic_mass_H * Gas_constant * TN )
```

Good luck,

Rich Younger

Subject: Re: Array multiplication: implicit loop query
Posted by george on Mon, 13 Aug 2001 09:59:11 GMT
View Forum Message <> Reply to Message

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 - 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.

George.

Subject: Re: Array multiplication: implicit loop query
Posted by John-David T. Smith on Mon, 13 Aug 2001 14:43:29 GMT
View Forum Message <> Reply to Message

```
Richard Younger wrote:
>
>  "Bill B." wrote:
>>
>>  george@apg.ph.ucl.ac.uk (george Millward) wrote in message
>
>>>  It
>>>  would seem that, to get this to work I need to make
>>>  Pres=fltarr(30,91,40).
>>
>>  Yes.
>>
>>  IDL> a = indgen(20,20)
>>  IDL> b = indgen(20)
>>  IDL> c = b * a
>>  IDL> help, c
>>  C               INT       = Array[20]
>>
>>  I believe you need to REPLICATE 'Pres' as needed.
>>
>
>  I've been converted to REBIN, myself.
>  (see the group archives for the dimensional juggling tutorial by JD
>  Smith this past spring)
>
```

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

P.S. Here's an example of this method's use in the field... a little function I cooked up to find where in one vector elements of another vector do *not* exist. As a bonus, not a histogram in there.

```
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
```