Subject: Re: The IDL way, summing variable sized slices of array.
Posted by Russell[1] on Wed, 04 Apr 2012 14:59:54 GMT
View Forum Message <> Reply to Message

On Apr 4, 4:00 am, d19997...@gmail.com wrote:
> Hi,
>
> I've recently been learning how to use REBIN/REFORM etc. to do the heavy lifting rather than loops, (saving me at least an order of magnitude in execution time in some of the code I'm working with). I have a situation now where I don't know if it's possible to completely remove loops so I was hoping someone more experienced could illuminate me.
>
> In essence the problem is that I have a 3D array which I want to reduce to a 2D array by summing over elements of the first dimension. This wouldn't be an issue apart from the fact that the range of elements that I wish to sum over varies depending on the value of the second dimension.
>
> In code what I have at the moment looks a bit like:
>
> d=DBLARR(nt,nl,ne) ;Array of data
> t=DBLARR(nt) ; Array of "axis" values
> b=DBLARR(nl,2) ;Array of summation limits
> p=DBLARR(nl,ne) ; Array of answer
>
> ;<<BIT OF CODE TO FILL THESE ARRAYS>>
>
> FOR i=0L,nl-1 DO BEGIN
>   tmp=TOTAL(d[b[i,0]:b[i,1],i,*],1) ;Sum elements
>   p[i,*]=tmp/TOTAL(t[b[i,0]:b[i,1]]) ;Store sum divided by sum of axis (i.e. get average value over summation range)
> ENDFOR
> RETURN,p
>
> Any help will be appreciated,
>
> Note whilst nl is not necessarily large in the cases I'll be looking at, i'm still interesting in "the IDL way" for this as part of my learning!
>
> Thanks,
> David

Not, sure... This sounds like the rare case were loops are useful.
BTW, I'm guessing you have a loop to fill the arrays? If so, then why
not stick this part in that loop?

Russell

Subject: Re: The IDL way, summing variable sized slices of array.
Posted by d19997919 on Wed, 04 Apr 2012 15:38:33 GMT
View Forum Message <> Reply to Message

On Wednesday, April 4, 2012 3:59:54 PM UTC+1, Russell wrote:

```
>  On Apr 4, 4:00 am, d19997...@gmail.com wrote:
>>  Hi,
>>
>>  I've recently been learning how to use REBIN/REFORM etc. to do the heavy lifting rather than loops, (saving me at least an order of magnitude in execution time in some of the code I'm working with). I have a situation now where I don't know if it's possible to completely remove loops so I was hoping someone more experienced could illuminate me.
>>
>>  In essence the problem is that I have a 3D array which I want to reduce to a 2D array by summing over elements of the first dimension. This wouldn't be an issue apart from the fact that the range of elements that I wish to sum over varies depending on the value of the second dimension.
>>
>>  In code what I have at the moment looks a bit like:
>>
>>  d=DBLARR(nt,nl,ne) ;Array of data
>>  t=DBLARR(nt) ; Array of "axis" values
>>  b=DBLARR(nl,2) ;Array of summation limits
>>  p=DBLARR(nl,ne) ; Array of answer
>>
>>  ;<<BIT OF CODE TO FILL THESE ARRAYS>>
>>
>>  FOR i=0L,nl-1 DO BEGIN
>>     tmp=TOTAL(d[b[i,0]:b[i,1],i,*],1)  ;Sum elements
>>     p[i,*]=tmp/TOTAL(t[b[i,0]:b[i,1]]) ;Store sum divided by sum of axis (i.e. get average value over summation range)
>>  ENDFOR
>>  RETURN,p
>>
>>  Any help will be appreciated,
>>
>>  Note whilst nl is not necessarily large in the cases I'll be looking at, i'm still interesting in "the IDL way" for this as part of my learning!
>>
>>  Thanks,
>>  David
>
>  Not, sure...  This sounds like the rare case were loops are useful.
>  BTW, I'm guessing you have a loop to fill the arrays?  If so, then why
>  not stick this part in that loop?
>
>  Russell
```

Actually I've managed to eliminate loops in the other bit of code to fill the arrays (and this code actually consists of a few other functions).

One way I've thought of doing this is to create a logical array which has the same dimensions as d and is zero outside the b index range and 1 inside. I can then multiply d by this and then just sum over the whole array, seen as d is now zero outside the region of interest it shouldn't effect the result of the call to TOTAL.

Any tips on the best way to achieve this?

Subject: Re: The IDL way, summing variable sized slices of array.
Posted by cgguido on Wed, 04 Apr 2012 16:01:37 GMT

View Forum Message <> Reply to Message

How about something like:
lo=rebin(b[*,0], [nl,nt])
hi=rebin(b[*,1], [nl,nt])

logic=(d ge lo) AND (d lt hi)
newd=d*logic

Not sure I get what your 't' array is for, but:

p=total(newd,1)/total(logic,1) ;should do it

I may very well have switched dimensions 7 times in the above code. columns-rows rows-columns... yikes

G

Subject: Re: The IDL way, summing variable sized slices of array.
Posted by d19997919 on Thu, 05 Apr 2012 08:39:37 GMT

View Forum Message <> Reply to Message

On Wednesday, April 4, 2012 5:01:37 PM UTC+1, Gianguido Cianci wrote:
> How about something like:
> lo=rebin(b[*,0], [nl,nt])
> hi=rebin(b[*,1], [nl,nt])
>
> logic=(d ge lo) AND (d lt hi)
> newd=d*logic
>
> Not sure I get what your 't' array is for, but:
>
> p=total(newd,1)/total(logic,1) ;should do it
>

> I may very well have switched dimensions 7 times in the above code. columns-rows rows-columns... yikes
>
> G

Thanks, your approach didn't quite work (probably due to my poor description of my problem) but something similar based of it did.

For interest the "solution" is given below, any comments on things I'm doing inefficiently would be appreciated.

```
<SOME INITIALISATION CODE/INTERFACE CODE>

;Get dimensions
nth=N_ELEMENTS(bmag)
nla=N_ELEMENTS(lambda)
nen=N_ELEMENTS(energy)

;Find where theta=+/- pi
lind=NEAREST(theta,-!DPI)
uind=NEAREST(theta,!DPI)

;Calculate arc length array
 dl=REBIN(GET_FIELDLINE_WEIGHT(THETA=THETA,JACOB=JACOB),[nth,
nla,nen],/SAMPLE)

;Precession drift is the bounce average of the grad-B+curvature drifts
;Define for passing particles as -1
;->Begin by getting the drift frequencies
;Pass in extra to allow ky and delt to be specified if desired
 drift=get_drift_freq(LAMBDA=LAMBDA,BMAG=BMAG,ENERGY=ENERGY,$
 CVDRIFT=CVDRIFT,GBDRIFT=GBDRIFT,_EXTRA=_EXTRA)

;Multiply drift frequency by arc length array
drift=TEMPORARY(drift)*dl

;Now want to bounce average this, so get the bounce points
 bounce=get_bounce_points(THETA=THETA,LAMBDA=LAMBDA,BMAG=BMAG )

;For passing particles, which have bounce == -1, set to indices
;corresponding to +/- Pi
bounce[*,0]=(lind+1)*(bounce[*,0] EQ -1)+TEMPORARY(bounce[*,0])
bounce[*,1]=(uind+1)*(bounce[*,1] EQ -1)+TEMPORARY(bounce[*,1])

;Create logic arrays about which regions we're interested in
 lo=TRANSPOSE(REBIN(bounce[*,0],[nla,nen,nth],/SAMPLE),[2,0,1 ])
 hi=TRANSPOSE(REBIN(bounce[*,1],[nla,nen,nth],/SAMPLE),[2,0,1 ])
logi=REBIN(BINDGEN(nth),[nth,nla,nen],/SAMPLE)
```

```
logi=(logi GT lo) AND (logi LT hi)

;Clear memory
bounce=0

;Now get velocity grids
 vel=get_velocity_grids(LAMBDA=LAMBDA,BMAG=BMAG,ENERGY=ENERGY )
vpa=REFORM(vel.vpa[*,*,*,0],/OVERWRITE)
;Set any vpa=0 points to 1, ok as we ignore these points anyway in the end
vpa=(vpa EQ 0)+TEMPORARY(vpa)

;Clear memory
vel=0

;Sum up arrays over whole theta dimension, noting that
;logi==0 outside the bounce points
drift=TOTAL((drift*logi/vpa),1)
div=TOTAL((dl*logi/vpa),1)

;Account for totally trapped particles where div==0
div=div+(div EQ 0)
drift=TEMPORARY(drift)*(div NE 0)+(div EQ 0)
precfreq=TEMPORARY(drift)/TEMPORARY(div)

;Clear memory
drift=0
div=0

;Change passing particles value if requested
IF N_ELEMENTS(PASSING) NE 0 THEN precfreq=precfreq*(lo[0,*,*] NE lind)+PASSING*(lo[0,*,*] EQ lind)

;Make and return answer
RETURN,precfreq
```