Subject: Re: arithmetic operation on array
Posted by Phillip Miller on Mon, 12 Aug 2013 22:51:57 GMT
View Forum Message <> Reply to Message

On 2013-08-12 21:24:07 +0000, David Fanning said:

```
> Phillip Miller writes:
>
>> Possibly a dumb question, but I'm pretty new to IDL:
>>
>> I have a geographically explicit time-series with 456 time steps and a
>> 1 degree resolution, so an array of dimensions 360 x 180 x 456, and I
>> would like to recalculate it as the anomaly from the time-series
>> average.
>>
>> I can calculate the time series average no problem
>>
>>> average = mean(data, dimension=3)
>>
>> But, of course, when I try
>>
>>> anomaly = data - mean(data, dimension=3)
>>
>> then I "lose" my third dimension, and end up with an array of 360 x
>> 180, rather than what I want, which is an array that is the same size
>> as my original.
>>
>> I know that I could loop it like
>>
>>> for i = 0,456 data[*,*,i] = data[*,*,i] - mean(data, dimension=3)
>>
>> but I feel like there must be a better way than making a for loop.  Am
>> I supposed to duplicate mean(data, dimension=3) times 456 in order to
>> create an identically sized array for the minus operation? (i.e., an
>> array with dimensions 360 x 180 x 456, but where each of the 456
>> "slices" is identical)
>>
>> Thanks in advance for any suggestions!
>
> The answer depends on how big your arrays are, how much on-board memory
> your machine has, and whether there is a coffee machine nearby.
> Personally, I wouldn't be worrying about optimizing your code until you
> discover there is a need to do so.
>
> I wouldn't, however, use code like this:
>
>    for i = 0,456 data[*,*,i] = data[*,*,i] - mean(data, dimension=3)
>
```

```
>  This is guaranteed to be slow, because you are calculating the mean
>  every time through the loop. Since that doesn't change, calculate it
>  once:
>
>     average = mean(data, dimension=3)
>     for i = 0,456 data[*,*,i] = data[*,*,i] - average
>
>  If you want to give it a try the IDL way, I would try something like
>  this:
>
>     average = mean(data, dimension=3)
>     data = Temporary(data) - Rebin(average, 360, 180, 456)
>
>  You can time it by wrapping the code in the routines TIC and TOC. We
>  would all be curious to see the results. :-)
>
>  Cheers,
>
>  David
```

Thanks, that was quite helpful. To satisfy your curiosity, I did some ticking and tocking, and I used a "fake" data set for the sake of this example so that I could try it with even more data and get a bigger difference in the time (double precision array with dimensions 360 x 180 x 1000). All of this is on an iMac with a 2.8 GHz i7 and 16 GB of RAM.

First of all, I wanted to see just how much slower my original for loop would be, but I had to make an adjustment because
> for i = 0,456 data[*,*,i] = data[*,*,i] - mean(data, dimension=3)
has a major flaw (aside from having forgotten the "do" and having my index end one-too-high for a dimension of size 456 starting at zero) which is that as my "data" array has a slice rewritten at each step through the for loop, the mean of all the slices gets incorrectly altered as well. So I had to create a new array to accept the "anomaly" values at each step in the for loop.

Anyway, the following code:

```
data = dindgen(360,180,1000)
; Example 1: Original (slow) loop
tic
anom = data
for i=0, 999 do anom[*,*,i] = data[*,*,i]-mean(data, dimension=3)
print, 'Example 1: Original (slow) loop'
toc
; Example 2: David's (better) loop
tic
```

```
average = mean(data, dimension=3)
for i=0, 999 do data[*,*,i] = data[*,*,i]-average
print, 'Example 2: David''s (better) loop'
toc
; Example 3: the IDL way
tic
average = mean(data, dimension=3)
data = temporary(data) - rebin(average,360,180,1000)
print, 'Example 3: the IDL way'
toc
```

gives the following result:

```
Example 1: Original (slow) loop
% Time elapsed: 599.96640 seconds.
Example 2: David's (better) loop
% Time elapsed: 0.83603501 seconds.
Example 3: the IDL way
% Time elapsed: 1.4728391 seconds.
```

So, not surprisingly, the original loop I had would have required a coffee machine near by after all. I knew that would be inefficient, but I had thought it due to the for loop in and of itself, rather than a poorly-written for loop that executes the exact same mean operation each iteration.

But quite interestingly, the "IDL way" is, in this particular case, actually slower than the more efficient for loop. Could that truly be, or am I doing something wrong here?

As an aside, I see why using temporary(data) in example 3 is more memory efficient. Would it have been appropriate to use it in example 2 as well, e.g.
for i=0, 999 do data[*,*,i] = temporary(data[*,*,i])-average
or would I be losing the original array as soon as the temporary function kicks in during the first iteration of the for loop?