Subject: arithmetic operation on array
Posted by Phillip Miller on Mon, 12 Aug 2013 20:59:16 GMT
View Forum Message <> Reply to Message

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!

Subject: Re: arithmetic operation on array
Posted by David Fanning on Mon, 12 Aug 2013 21:24:07 GMT
View Forum Message <> Reply to Message

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

--
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: arithmetic operation on array
Posted by Phillip Bitzer on Mon, 12 Aug 2013 22:22:14 GMT

View Forum Message <> Reply to Message

OK, I'll bite. There are three ways I can think to do this off the top of my head:
1) Do a loop, like Phillip said (what a fantastic name :-))
2) Rebin, like David said
3) Use the mysterious "add an extra dimension" method (
https://groups.google.com/d/msg/comp.lang.idl-pvwave/Vu9rzqc kBNQ/HvkK_QnJrsgJ and more recently https://groups.google.com/d/msg/comp.lang.idl-pvwave/dM8XXas Eio0/d3__pvX7svMJ)

Here are the sample code I used:

```
data = RANDOMU(1L, 360, 180, 456)
avg = MEAN(data, DIM=3)
mData1 = FLTARR(360, 180, 456)

tic & for i=0, 455 do mData1[*,*,i] = data[*,*,i] - avg & toc

mData2 = FLTARR(360, 180, 456)

tic & mData2 = data - Rebin(avg, 360, 180, 456) & toc
tic & data = TEMPORARY(data) - Rebin(avg, 360, 180, 456) & toc ;just for completeness

data = RANDOMU(1L, 360, 180, 456) ;redefine data - we changed it above
mData3 = FLTARR(360, 180, 456)

tic & mData3 = data - avg[*, *, 0] & toc
```

I used mData (modified data) arrays so I can check I get the same answer, regardless of the method.

The four times I get are, in order relative to the above:

% Time elapsed: 0.12749481 seconds.
% Time elapsed: 0.33231401 seconds.
% Time elapsed: 0.33304191 seconds.
% Time elapsed: 0.019619942 seconds.

So, it seems the mysterious extra dimension method is the fastest, by an order of magnitude. Whoa.

For prosperity,
IDL> print, !VERSION
{ x86_64 darwin unix Mac OS X 8.2.2 Jan 23 2013   64   64}

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?

Subject: Re: arithmetic operation on array
Posted by Phillip Miller on Mon, 12 Aug 2013 23:08:38 GMT
View Forum Message <> Reply to Message

On 2013-08-12 22:22:14 +0000, Phillip Bitzer said:

```
> OK, I'll bite. There are three ways I can think to do this off the top
> of my head:
> 1) Do a loop, like Phillip said (what a fantastic name :-) )
> 2) Rebin, like David said
> 3) Use the mysterious "add an extra dimension" method
> ( https://groups.google.com/d/msg/comp.lang.idl-pvwave/Vu9rzqc kBNQ/HvkK_QnJrsgJ
> and more recently
>  https://groups.google.com/d/msg/comp.lang.idl-pvwave/dM8XXas Eio0/d3__pvX7svMJ)
>
>
> Here are the sample code I used:
>
> data = RANDOMU(1L, 360, 180, 456)
> avg = MEAN(data, DIM=3)
> mData1 = FLTARR(360, 180, 456)
>
> tic & for i=0, 455 do mData1[*,*,i] = data[*,*,i] - avg & toc
>
> mData2 = FLTARR(360, 180, 456)
>
> tic & mData2 = data - Rebin(avg, 360, 180, 456) & toc
> tic & data = TEMPORARY(data) - Rebin(avg, 360, 180, 456) & toc ;just
> for completeness
>
> data = RANDOMU(1L, 360, 180, 456) ;redefine data - we changed it above
> mData3 = FLTARR(360, 180, 456)
>
> tic & mData3 = data - avg[*, *, 0] & toc
>
> I used mData (modified data) arrays so I can check I get the same
> answer, regardless of the method.
>
> The four times I get are, in order relative to the above:
> % Time elapsed: 0.12749481 seconds.
> % Time elapsed: 0.33231401 seconds.
> % Time elapsed: 0.33304191 seconds.
> % Time elapsed: 0.019619942 seconds.
>
> So, it seems the mysterious extra dimension method is the fastest, by
```

> an order of magnitude. Whoa.
>
> For prosperity,
> IDL> print, !VERSION
> { x86_64 darwin unix Mac OS X 8.2.2 Jan 23 2013   64   64}

Excellent, thanks for the tip.
I tried it on my 360x180x1000 double precision array and got the
similar results as you, though not quite an order of magnitude
difference between it and the "improved" loop method, the mysterious
extra dimension method was indeed the fastest

Subject: Re: arithmetic operation on array
Posted by Phillip Miller on Mon, 12 Aug 2013 23:17:58 GMT
View Forum Message <> Reply to Message

On 2013-08-12 23:08:38 +0000, Phillip Miller said:

```
>  On 2013-08-12 22:22:14 +0000, Phillip Bitzer said:
>
>>  OK, I'll bite. There are three ways I can think to do this off the top
>>  of my head:
>>  1) Do a loop, like Phillip said (what a fantastic name :-) )
>>  2) Rebin, like David said
>>  3) Use the mysterious "add an extra dimension" method
>>  ( https://groups.google.com/d/msg/comp.lang.idl-pvwave/Vu9rzqc kBNQ/HvkK_QnJrsgJ
>>  and more recently
>>   https://groups.google.com/d/msg/comp.lang.idl-pvwave/dM8XXas Eio0/d3__pvX7svMJ)
>>
>>
>>  Here are the sample code I used:
>>
>>  data = RANDOMU(1L, 360, 180, 456)
>>  avg = MEAN(data, DIM=3)
>>  mData1 = FLTARR(360, 180, 456)
>>
>>  tic & for i=0, 455 do mData1[*,*,i] = data[*,*,i] - avg & toc
>>
>>  mData2 = FLTARR(360, 180, 456)
>>
>>  tic & mData2 = data - Rebin(avg, 360, 180, 456) & toc
>>  tic & data = TEMPORARY(data) - Rebin(avg, 360, 180, 456) & toc ;just
>>  for completeness
>>
>>  data = RANDOMU(1L, 360, 180, 456) ;redefine data - we changed it above
>>  mData3 = FLTARR(360, 180, 456)
>>
```

>> tic & mData3 = data - avg[*, *, 0] & toc
>>
>> I used mData (modified data) arrays so I can check I get the same
>> answer, regardless of the method.
>>
>> The four times I get are, in order relative to the above:
>> % Time elapsed: 0.12749481 seconds.
>> % Time elapsed: 0.33231401 seconds.
>> % Time elapsed: 0.33304191 seconds.
>> % Time elapsed: 0.019619942 seconds.
>>
>> So, it seems the mysterious extra dimension method is the fastest, by
>> an order of magnitude. Whoa.
>>
>> For prosperity,
>> IDL> print, !VERSION
>> { x86_64 darwin unix Mac OS X 8.2.2 Jan 23 2013   64   64}
>
> Excellent, thanks for the tip.
> I tried it on my 360x180x1000 double precision array and got the
> similar results as you, though not quite an order of magnitude
> difference between it and the "improved" loop method, the mysterious
> extra dimension method was indeed the fastest

Hmm, perhaps I spoke too soon. The result of the extra dimension
method has lost the third dimension.
After I run that code, I get
IDL> help, mdata3
MDATA3   FLOAT   = Array[360, 180]

Subject: Re: arithmetic operation on array
Posted by David Fanning on Mon, 12 Aug 2013 23:18:49 GMT
View Forum Message <> Reply to Message

Phillip Bitzer writes:

> Here are the sample code I used:
>
> data = RANDOMU(1L, 360, 180, 456)
> avg = MEAN(data, DIM=3)
> mData1 = FLTARR(360, 180, 456)
>
> tic & for i=0, 455 do mData1[*,*,i] = data[*,*,i] - avg & toc
>
> mData2 = FLTARR(360, 180, 456)
>
> tic & mData2 = data - Rebin(avg, 360, 180, 456) & toc

```
>  tic & data = TEMPORARY(data) - Rebin(avg, 360, 180, 456) & toc ;just for completeness
>
>  data = RANDOMU(1L, 360, 180, 456) ;redefine data - we changed it above
>  mData3 = FLTARR(360, 180, 456)
>
>  tic & mData3 = data - avg[*, *, 0] & toc
>
```
> I used mData (modified data) arrays so I can check I get the same answer, regardless of the method.
>
> The four times I get are, in order relative to the above:

```
>  % Time elapsed: 0.12749481 seconds.
>  % Time elapsed: 0.33231401 seconds.
>  % Time elapsed: 0.33304191 seconds.
>  % Time elapsed: 0.019619942 seconds.
```
>
> So, it seems the mysterious extra dimension method is the fastest, by an order of magnitude. Whoa.

Well, running your code, I find this:

```
 Elapsed Time:  0.168000
 Elapsed Time:  0.376000
 Elapsed Time:  0.347000
 Elapsed Time:  0.009000
```

But, that last number is truly unbelievable. A quick Help on mData3
reveals this:

```
IDL> help, mdata3
MDATA3          FLOAT     = Array[360, 180]
```

Whoops! I think you did one subtraction, not 456. :-)

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: arithmetic operation on array

Posted by Phillip Bitzer on Mon, 12 Aug 2013 23:37:12 GMT
View Forum Message <> Reply to Message

>
> Whoops! I think you did one subtraction, not 456. :-)
>

Aack! I swear I checked that. The dog must have ate the extra dimension :-P

I mean, really, what's 455 missing operations between friends?

Sorry for this detour down TripleCheckYourWork Lane. Everyone back to the IDL Freeway now....

Subject: Re: arithmetic operation on array
Posted by John Correira on Wed, 14 Aug 2013 19:24:17 GMT
View Forum Message <> Reply to Message

On 08/12/2013 06:51 PM, Phillip Miller wrote:
> 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?
>

By subscripting the data array you are making a copy of it, so using
TEMPORARY won't help.

John

Subject: Re: arithmetic operation on array
Posted by wlandsman on Wed, 14 Aug 2013 19:54:26 GMT
View Forum Message <> Reply to Message

While TEMPORARY() won't help things, you can improve the speed by not using asterisks on the left side of the assignment statement.   instead write

for i=0, 999 do data[0,0,i] = temporary(data[*,*,i])-average

http://www.idlcoyote.com/code_tips/asterisk.html

--Wayne

On Wednesday, August 14, 2013 3:24:17 PM UTC-4, John Correira wrote:
> On 08/12/2013 06:51 PM, Phillip Miller wrote:

>
>> 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?
>
>>
>
>
>
> By subscripting the data array you are making a copy of it, so using
>
> TEMPORARY won't help.
>
>
>
> John