Subject: Re: Using where() on slices of data cubes
Posted by David Fanning on Tue, 20 Oct 2009 13:03:22 GMT
View Forum Message <> Reply to Message

Conor writes:

> I feel like this should be an easy one, but I've never quite figured
> it out. Let's say I got a data cube and I want to do something on
> just a slice of it, say I want to turn certain values in a column into
> something else:
>
> w = where(cube[1,*,*] lt 0)
>
> It seems like you should be able to do something like this:
>
> cube[1,w] = 1e24
>
> But that doesn't work... Somehow I can't quite figure out the right
> way to do this.

It *seems* like there should be a simple way to do this,
but if there is, I haven't found it. What would make
sense to me is this:

```
(cube[1,*,*])[w] = 1e24
```

But this gives the error message that you can't store into
a temporary variable. (The ol' pass by reference/pass by value
thing, I suppose.)

I have always resorted to making a subset of the data, like
this:

```
sub = cube[1,*,*]
w = where(sub lt 0)
sub[w] = 1e24
cube[1,*,*] = sub
```

Not at all elegant, I admit. :-(

Cheers,

David

--
David Fanning, Ph.D.

Subject: Re: Using where() on slices of data cubes
Posted by Conor on Tue, 20 Oct 2009 13:20:47 GMT
View Forum Message <> Reply to Message

Well it works, so I can't knock it that much :) I had resorted to
using a for loop :(

On Oct 20, 9:03 am, David Fanning <n...@dfanning.com> wrote:
> Conor writes:
>> I feel like this should be an easy one, but I've never quite figured
>> it out. Let's say I got a data cube and I want to do something on
>> just a slice of it, say I want to turn certain values in a column into
>> something else:
>
>> w = where(cube[1,*,*] lt 0)
>
>> It seems like you should be able to do something like this:
>
>> cube[1,w] = 1e24
>
>> But that doesn't work... Somehow I can't quite figure out the right
>> way to do this.
>
> It *seems* like there should be a simple way to do this,
> but if there is, I haven't found it. What would make
> sense to me is this:
>
> (cube[1,*,*])[w] = 1e24
>
> But this gives the error message that you can't store into
> a temporary variable. (The ol' pass by reference/pass by value
> thing, I suppose.)
>
> I have always resorted to making a subset of the data, like
> this:
>
> sub = cube[1,*,*]
> w = where(sub lt 0)
> sub[w] = 1e24
> cube[1,*,*] = sub
>
> Not at all elegant, I admit. :-(
>

> Cheers,
>
> David
>
> --
> David Fanning, Ph.D.
> Fanning Software Consulting, Inc.
> Coyote's Guide to IDL Programming:http://www.dfanning.com/
> Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Using where() on slices of data cubes
Posted by greg.addr on Tue, 20 Oct 2009 14:20:46 GMT
View Forum Message <> Reply to Message

On 20 Okt., 14:22, Conor <cmanc...@gmail.com> wrote:
> I feel like this should be an easy one, but I've never quite figured
> it out. Let's say I got a data cube and I want to do something on
> just a slice of it, say I want to turn certain values in a column into
> something else:
>
> w = where(cube[1,*,*] lt 0)
>
> It seems like you should be able to do something like this:
>
> cube[1,w] = 1e24
>
> But that doesn't work... Somehow I can't quite figure out the right
> way to do this.

This method avoids the need for any duplication:

```
w=where(cube[1,*,*] lt 0)
sz=size(cube,/dim)
cube=reform(cube,[sz[0],sz[1]*sz[2]],/overwrite)
cube[1,w]=1e24
cube=reform(cube,sz,/overwrite)
```

regards,
Greg

Subject: Re: Using where() on slices of data cubes
Posted by Foldy Lajos on Tue, 20 Oct 2009 14:26:52 GMT
View Forum Message <> Reply to Message

On Tue, 20 Oct 2009, Conor wrote:

> I feel like this should be an easy one, but I've never quite figured
> it out. Let's say I got a data cube and I want to do something on
> just a slice of it, say I want to turn certain values in a column into
> something else:
>
> w = where(cube[1,*,*] lt 0)
>
> It seems like you should be able to do something like this:
>
> cube[1,w] = 1e24
>
> But that doesn't work... Somehow I can't quite figure out the right
> way to do this.
>

cube[where(cube[1,*,*] lt 0)*(size(cube, /dim))[0]+1] = 1e24

regards,
lajos

Subject: Re: Using where() on slices of data cubes
Posted by lecacheux.alain on Tue, 20 Oct 2009 14:34:34 GMT
View Forum Message <> Reply to Message

> This method avoids the need for any duplication:
>
> w=where(cube[1,*,*] lt 0)
> sz=size(cube,/dim)
> cube=reform(cube,[sz[0],sz[1]*sz[2]],/overwrite)
> cube[1,w]=1e24
> cube=reform(cube,sz,/overwrite)
>
> regards,
> Greg

Is'nt this more efficient ?
cube[1,*,*] = 1e24*(cube[1,*,*] lt 0) + cube[1,*,*]*(cube[1,*,*] ge
0)

alx.

Subject: Re: Using where() on slices of data cubes
Posted by penteado on Tue, 20 Oct 2009 14:39:18 GMT

View Forum Message <> Reply to Message

On Oct 20, 10:22 am, Conor <cmanc...@gmail.com> wrote:
> I feel like this should be an easy one, but I've never quite figured
> it out. Let's say I got a data cube and I want to do something on
> just a slice of it, say I want to turn certain values in a column into
> something else:
>
> w = where(cube[1,*,*] lt 0)
>
> It seems like you should be able to do something like this:
>
> cube[1,w] = 1e24
>
> But that doesn't work... Somehow I can't quite figure out the right
> way to do this.

It does not apply to this case, but for future reference, it may be
useful to know that it could be done directly if the slice was the
other way around (a subset of indices on the left):

```
w=where(cube[*,*,i] lt 0)
nel=n_elements(cube[*,*,i])
cube[i*nel+w]=1e24
```

Subject: Re: Using where() on slices of data cubes
Posted by Paul Van Delst[1] on Tue, 20 Oct 2009 15:00:32 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
> Conor writes:
>
>> I feel like this should be an easy one, but I've never quite figured
>> it out. Let's say I got a data cube and I want to do something on
>> just a slice of it, say I want to turn certain values in a column into
>> something else:
>>
>> w = where(cube[1,*,*] lt 0)
>>
>> It seems like you should be able to do something like this:
>>
>> cube[1,w] = 1e24
>>
>> But that doesn't work... Somehow I can't quite figure out the right
>> way to do this.
>
> It *seems* like there should be a simple way to do this,

> but if there is, I haven't found it. What would make
> sense to me is this:
>
> (cube[1,*,*])[w] = 1e24
>
> But this gives the error message that you can't store into
> a temporary variable. (The ol' pass by reference/pass by value
> thing, I suppose.)

Is it possible at all to use IDL pointer for aliasing? E.g. in Fortran I would do something like the following

```
 real, target  :: cube(:,:,:)
 real, pointer :: alias(:,:) => null()

 alias => cube(1,:,:)

 where ( alias < 0.0 )
   alias = 1e24
 end where

 nullify(alias)
```

Is there an IDL equivalent to the "alias => cube(1,:,:)" statement in Fortran?

If there is, then that might get around the tired old "can't store into a temporary variable" issue that plagues IDL due to it sticking to pass by reference only. (Man, I wish they would fix that.)

The only way I can figure out to do the aliasing ends up copying the data.

Anyway...

cheers,

paulv

Subject: Re: Using where() on slices of data cubes
Posted by greg.addr on Tue, 20 Oct 2009 16:18:43 GMT

View Forum Message <> Reply to Message

> Is'nt this more efficient ?
> cube[1,*,*] = 1e24*(cube[1,*,*] lt 0) + cube[1,*,*]*(cube[1,*,*] ge
> 0)

I reckon you're copying out the slice twice, processing through it 5

times, and copying it back. That's compared to one scan through the slice and one indexed substitution (possibly many fewer items than one slice). The 'reform' with /overwrite costs nothing.

regards,
Greg

Subject: Re: Using where() on slices of data cubes
Posted by David Fanning on Tue, 20 Oct 2009 20:03:29 GMT
View Forum Message <> Reply to Message

Lajos writes:

>
> On Tue, 20 Oct 2009, Conor wrote:
>
>> I feel like this should be an easy one, but I've never quite figured
>> it out. Let's say I got a data cube and I want to do something on
>> just a slice of it, say I want to turn certain values in a column into
>> something else:
>>
>> w = where(cube[1,*,*] lt 0)
>>
>> It seems like you should be able to do something like this:
>>
>> cube[1,w] = 1e24
>>
>> But that doesn't work... Somehow I can't quite figure out the right
>> way to do this.
>>
>
>
> cube[where(cube[1,*,*] lt 0)*(size(cube, /dim))[0]+1] = 1e24

Humm. Well, I suppose, for this *specific* problem. I can't wait to see the general solution, though, for a slice in any direction. Do you suppose it will fit on a page of paper? :-)

Cheers,

David
--
David Fanning, Ph.D.
Coyote's Guide to IDL Programming (www.dfanning.com)
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Using where() on slices of data cubes
Posted by JDS on Tue, 20 Oct 2009 20:03:30 GMT
View Forum Message <> Reply to Message

On Oct 20, 8:22 am, Conor <cmanc...@gmail.com> wrote:
> I feel like this should be an easy one, but I've never quite figured
> it out. Let's say I got a data cube and I want to do something on
> just a slice of it, say I want to turn certain values in a column into
> something else:
>
> w = where(cube[1,*,*] lt 0)
>
> It seems like you should be able to do something like this:
>
> cube[1,w] = 1e24
>
> But that doesn't work... Somehow I can't quite figure out the right
> way to do this.

It's impossible for IDL to know the dimensionality of the original array which from which you extracted a given set of elements. WHERE simply gives you the zero-based running index into a given pile of data, independent of its form and (much less) the form of the parent array from which it was derived. So it's not at all surprising that the returned set of indices doesn't "plug right in" in some convenient way. The (likely) quickest solution in this case is the REFORM,/OVERWRITE method given by Greg, since it doesn't actually copy any data. But unfortunately, it's not at all general. For example, suppose you had been interested in longitudinal slices instead, ala:

```
w=where(cube[*,1,*] lt 0)
```

A solution similar to Greg's would require first TRANSPOSE'ing the cube such that the elements of the slice of interest could be put in order along some dimension, e.g.:

```
w=where(cube[*,1,*] lt 0)
sz=size(cube,/dim)
cube=reform(transpose(cube,[0,2,1]),[sz[0]*sz[2],sz[1]])
cube[w,1]=1e24
 cube=transpose(reform(cube,sz[0],sz[2],sz[1],/overwrite),[0, 2,1])
```

This obviously creates two transposed copies of the cube, which is memory and CPU inefficient, not to mention difficult to remember. For operating on small cubes, or a large fraction of the cube elements, it might be reasonable. If, however, you have only a few elements to access in a very large array (say just a sprinkling of negatives, in your example), this would be very wasteful indeed.

In my opinion, a far better general solution to these sorts of problems is to master the ability to recreate *yourself* any of the index computations IDL does for you (and, by extension, any that it doesn't). For example, when you write

```
slice = cube[1,*,*]
```

IDL doesn't just conjure the appropriate elements out of thin air. It examines the dimensions of 'cube', and implicitly constructs a one-dimensional index vector for all of the indices being referenced. This happens in the background, but you can easily verify it by seeing how much memory IDL has used for this temporary index vector. As a side note, this can occasionally be memory inefficient, which is why it's sometimes *preferable* to construct your own indices, even when IDL could have done it for you (see http://www.dfanning.com/misc_tips/submemory.html).

As for the posed problem, let's skip to the end. In your example, the answer looks like:

```
sz=size(cube,/DIMEN)
cube[1+sz[0]*(w mod sz[1] + w/sz[1]*sz[1])] = 1e24
```

Where does this come from? Your original cube "slice" has dimensions [sz[1],sz[2]], and is at an x position of 1. Recall that

```
slice_col = w mod sz[1]
```

gives the column inside this slice, and

```
slice_row = w/sz[1]
```

gives the row. Nothing fancy there. If you forget these, you can easily use the IDL provided convenience function ARRAY_INDICES instead:

```
slice_col_row = array_indices(sz[1:2],w,/DIMENSIONS)
```

However, this is just doing the same pair of computations, which are not that hard to remember.

So far so good. We need to create a one dimensional index vector appropriate for the cube's dimensionality, ala (very generally):

```
ind = x + sx * (y + sy * z)
```

Now comes the (slightly) tricky bit. We need to do this *for the selected slice elements*. But wait! The slice's column (x direction) is the full cube's row (aka y direction), and the slice's row is the

full cube's plane (aka z direction). So what we need looks like:

```
ind = x + sx * (slice_col + sy * slice_row)
```

aka

```
ind = 1 + sz[0] * (w mod sz[1] + w/sz[1]*sz[1])
```

One other wrinkle. Notice I didn't write sz[1] * w/sz[1]. This is because operator precedence would compute this as (sz[1]*w)/sz[1], which would give you.... w -- not what you want. You can either write sz[1]*(w/sz[1]), or just rearrange terms as I have done.

OK, you may be asking yourself, what has this really gained me? A lot, actually. Once you become fluent in converting back and forth between one-dimensional indices and [x,y,z,...] index sets in arbitrary dimensions, you are able to manipulate with ease the full range of indexing problems, and are, most importantly, no longer reliant on IDL's convenient but limited higher-order indexing operators. For example, suppose I had originally issued the call:

```
w=where(cube[1,5:*,10:1024] lt 0)
```

you should easily be able to generalize the above arguments to access these elements.

Subject: Re: Using where() on slices of data cubes
Posted by David Fanning on Tue, 20 Oct 2009 20:32:35 GMT
View Forum Message <> Reply to Message

JD Smith writes:

> you should easily be able to generalize the above arguments to access
> these elements

I think in this case the word "easily" might be
too subtly sarcastic to be easily appreciated by
the vast majority of this newsgroup. :-)

Cheers,

David

--
David Fanning, Ph.D.
Coyote's Guide to IDL Programming (www.dfanning.com)
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Using where() on slices of data cubes
Posted by JDS on Wed, 21 Oct 2009 21:52:45 GMT
View Forum Message <> Reply to Message

On Oct 20, 4:32 pm, David Fanning <n...@dfanning.com> wrote:
> JD Smith writes:
>> you should easily be able to generalize the above arguments to access
>> these elements
>
> I think in this case the word "easily" might be
> too subtly sarcastic to be easily appreciated by
> the vast majority of this newsgroup. :-)

(Almost) no sarcasm was intended.

Suppose you have this:

```
w=where(cube[1,5:*,10:1024] lt 0)
```

The "slice" is no longer as large as the cube in the yz dimensions,
and is offset by [5,10] too. So

```
y_full_cube = slice_column + 5
z_full_cube = slice_row    + 10
```

and since the slice is smaller than the cube by 5 columns, to convert
our WHERE index vector w into col,row in the slice, we use

```
slice_column = w mod (sz[1]-5)
slice_row    = w/(sz[1]-5)
```

Putting it all together we have:

```
ind = 1 + sz[0] * (5 + w mod (sz[1]-5) + (10 + w/(sz[1]-5)) * sz[1])
```

JD