Subject: Re: IDL FOR Loop variable increments
Posted by Jeremy Bailin on Tue, 23 Sep 2008 14:48:46 GMT
View Forum Message <> Reply to Message

On Sep 22, 11:54 am, Raghu <raghuram.narasim...@gmail.com> wrote:
> On Sep 21, 3:37 pm, Raghu <raghuram.narasim...@gmail.com> wrote:
>
>
>
>> On Sep 21, 6:44 am, Bulrush <Wasit.Weat...@gmail.com> wrote:
>
>>> On Sep 20, 12:51 am, Raghu <raghuram.narasim...@gmail.com> wrote:
>
>>>> On Sep 19, 12:09 pm, pgri...@gmail.com wrote:
>
>>>> > R.G. Stockwell wrote:
>>>> > > "Jean H" <jghas...@DELTHIS.ucalgary.ANDTHIS.ca> wrote in message
>>>> > >news:gauiil$u32$1@news.ucalgary.ca...
>>>> > > ...
>>>> > > > Could you comment on the "risk" of changing the loop counter within the
>>>> > > > loop?
>
>>>> > > my 2 cents.
>
>>>> > > First, it is in changing the counter of a for loop.
>>>> > > A for loop explicitly outlines what all counter variables will be.
>
>>>> > > There are two things:
>
>>>> > > 1) infinite loop, one could easily change the counter to never
>>>> > > reach the end condition. A (valid) for loop will always reach the end
>>>> > > condition.
>
>>>> > > 2) more insidious, you could inadvertantly cast the counter to a float from
>>>> > > an int, and then have one extra (and unintended) statement executed.
>
>>>> > This seems not to be possible in IDL, as loop counters, unlike normal
>>>> > variables, cannot change their type.
>
>>>> > Ciao,
>>>> > Paolo
>
>>>> > > instead of 0,1,2,3,4,5,6 (and not executing i = 7) you could get
>>>> > > 0,1,2,3,4,4.99999999,5.99999,6.99999999, (and effectively executing the
>>>> > > extra i ~ 7 step).
>
>>>> > > Cheers,
>>>> > > bob

>
>>>> Hi all,
>
>>>> Thanks for your replies. Just as David mentioned in his first
>>>> response, a while loop worked out much better. Within a single while
>>>> loop, i was able to accomplish the task, albeit a bit slowly because
>>>> of the non-array operation.
>
>>>> Thanks !- Hide quoted text -
>
>>>> - Show quoted text -
>
>>> Why not you do not share your final results with us to close this
>>> post.
>>> Elkunn
>
>> Hi,
>
>> I will. I don't have the code with me this weekend. I'll post it on
>> Monday at work.
>
>> Thanks,
>> Raghu
>
> Hello,
>
> here is the while loop piece of code that works now.
>
> b=0
> k=1
>
> while (b+k lt nb) do begin
>
> if finite(ndvislice[s,b+k]) eq 0 or finite (ndsislice[s,b+k]) eq 0
> then begin
> ndvi[s,b+k]=mask[s,r]
> ndsi[s,b+k]=mask[s,r]
> k=k+1
> endif else begin
> if (ndvislice[s,b+k] lt ndvislice[s,b])then begin;
> ndvi[s,b+k]=mask[s,r]
> ndsi[s,b+k]=mask[s,r]
> k=k+1
> endif else begin
> ndvi[s,b+k]=ndvislice[s,b+k]
> ndsi[s,b+k]=ndsislice[s,b+k]
> b=b+k
> k=1

> endelse
> endelse
> endwhile
>
> So, with two conditions, i can change the number of bands (b) or i can
> change the counter (k) depending on which condition is satisfied
> during an iteration. Although it works, it does take some time which
> might suggest that an array oriented code might work faster. But
> anyway, it works.
>
> Thanks!
>
> Raghu

Okay, I think I finally understand what you're trying to do. Let me
try to encapsulate it in words:

As long as ndvislice is not dropping (with NaNs treated as missing
data), ndvi and ndsi get set to ndvislice and ndsislice respectively,
otherwise they get set to the mask value.
(and ndvi[0] and ndsi[0] are unchanged from the input)

In which case, I think that this is a vectorized version:

```
; mask out all values at first, and only set the relevant ones
ndvi[s,1:*] = mask[s,r]
ndsi[s,1:*] = mask[s,r]
; ignore missing data
wherefinite = where(finite(ndvislice[s,*]) ne 0 and
finite(ndsislice[s,*]) ne 0, nfinite)
if nfinite gt 0 then begin
  ; which ones are greater than or equal to previous good value?
  wherenodrop = where(ndvislice[s,wherefinite[1:*]] ge
ndvislice[s,wherefinite], nnodrop)
  if nnodrop gt 0 then begin
    ; get the index back into the original array
    nodrop_i = wherefinite[wherenodrop]+1
    ndvi[s,nodrop_i] = ndvislice[s,nodrop_i]
    ndsi[s,nodrop_i] = ndsislice[s,nodrop_i]
  endif
endif
```

-Jeremy.