Subject: Re: rebin and !values.f_nan
Posted by Dick Jackson on Mon, 16 Jul 2007 07:03:19 GMT
View Forum Message <> Reply to Message

Hi all,

"Nick" <jungbinusu@hotmail.com> wrote in message
news:1184534401.344733.293590@i38g2000prf.googlegroups.com.. .
> Thank you for your reply.
> I could find where is 'NaN' vaule, but I had to omit these 'NaN' value
> in order to change array. It's hard point for me. If I change NaN
> values to zero, values which are changed by Rebin will be lower than
> original values (True values).
>
> A = [1.5,2.5,3.6,4,7,8.8,9.0,!values.f_nan]
> print, rebin(A, 1)
> ;True value is 5.2
> ;But if I changed NaN to zero, the result is 4.55
>
> The example case is so simple that I can fix it easily, but my data
> have 1440*720 array.
> So I couldn't fix thise one by one. Is there any methods?
> Thanks, Nick.

Just to clear up one possible confusion, IDL's Rebin() does handle each resulting bin separately, so any bin with no NaNs comes up fine:

IDL> A = [1.5,2.5,3.6,4,7,8.8,9.0,!values.f_nan]
IDL> print,rebin(a,4)
2.00000 3.80000 7.90000 NaN

Rebin() gives a NaN result if any value in the bin is NaN. Now, what I think Nick wants is for each bin to receive the mean of the finite values (the non-NaNs, or NaNaNs! or perhaps Ns... :-) that are present. A reasonable request, in the spirit of Mean(array, /NaN). I don't think there's a built-in way to do this NaN-tolerant Rebin, so here's my attempt.

I take the first dimension (1440) and split it into two dimensions (4, 360). I take the second dimension (720) and split it into two more dimensions (4, 180). Then I use Total(/NaN) to squash out the extra dimensions, getting the total of the desired values in each resulting array element. A similar Total-ing on the count of Finite() values gives the count of values summed for each result. Divide each total by each count and you get the mean of each bin's finite values. If a bin had all NaNs, the result should be NaN as well.

PRO RebinNaNTest

```
a = FIndGen(1440,720)                ; Sample data

a[1] = !values.f_nan                 ; to make one element NaN
;a[*,0:2] = !values.f_nan             ; to make three rows NaN
;a[*,0:3] = !values.f_nan             ; to make four rows NaN

Print, 'Rebin method:'

rebinResult = Rebin(a,360,180)

Print, rebinResult[0:1, 0:1]

b = Reform(a, 4, 360, 4, 180)          ; Make separate 'b' array in
                                       ; case you want to see this:
;print,total(a[0:3,0:3])
;          NaN
;print,total(a[0:3,0:3],/NaN)
;      34583.0
;print,total(b[*,0,*,0],/NaN)
;      34583.0
;;    But in practice, you could reform 'a' in place,
;;    which saves memory and is very fast:
;;    a = Reform(a, 4, 360, 4, 180, /Overwrite)

Print
Print, 'RebinNaN method:'

sumFinite = Total(Total(b, 3, /NaN), 1, /NaN)
nFinite = Total(Total(Finite(b), 3, /NaN), 1, /NaN)
result = sumFinite/nFinite

Print, result[0:1, 0:1]

END
```

With one NaN this seems to work:

```
Rebin method:
         NaN      2165.50
     7921.50      7925.50

RebinNaN method:
     2305.53      2165.50
     7921.50      7925.50
```

With three rows of NaNs:

```
IDL> rebinnantest
Rebin method:
         NaN         NaN
     7921.50     7925.50

RebinNaN method:
     4321.50     4325.50
     7921.50     7925.50
```

With four rows of NaNs:

```
IDL> rebinnantest
Rebin method:
         NaN         NaN
     7921.50     7925.50

RebinNaN method:
        -NaN        -NaN
     7921.50     7925.50
Program caused arithmetic error: Floating illegal operand
```

Rough timing test shows this takes 5-10 times as long as Rebin(), but it works!

This could be put into a nice general function to replace Rebin when you need NaN handling, but I'll see whether anyone pops up Monday morning with one already written!

Hope this helps.

--
Cheers,
-Dick

--
Dick Jackson Software Consulting http://www.d-jackson.com
Victoria, BC, Canada +1-250-220-6117 dick@d-jackson.com