Subject: Re: BYTSCL and NAN keyword
Posted by JD Smith on Tue, 02 Mar 2004 21:32:00 GMT
View Forum Message <> Reply to Message

On Tue, 02 Mar 2004 10:26:19 -0700, Kenneth Bowman wrote:

> In article <MPG.1aae6fc4ef56f7829896c3@news.frii.com>,
> David Fanning <david@dfanning.com> wrote:
>
>> Well, NANs are definitely floats (that is, they have a float-type
>> bit pattern). So it seems reasonable to me that if you are
>> trying to stuff it into a byte (which can only have values between
>> 0 and 255, that 0 is a good choice. (The only possible other
>> choice is 255, but that just turns your problem on its head.)
>>
>> I think what I would do is locate the NANs and save their
>> indices. Then scale my data into 255 colors (0 to 254), leaving
>> color index 255 for the NAN color (whatever that is). This will
>> take a couple of steps, but that's what IDL programs are for. :-)
>
> Right, I understand all that. I'm just trying to understand what the
> NAN keyword is good for.
>
> If you could say NAN = 255 (or 19, or whatever), instead of just /NAN,
> it would be of some use. That way the value that NANs become in the
> output could be distinguished from valid data.
>
> As it is, the NAN keyword is only of use if you are happy with NANs
> turning into valid data in the output (i.e., indistinguishable from
> small valid values).
>
> I have to consider this an error in the way the handling of NANs was
> implemented in BYTSCL.

```
IDL> print,bytscl([!VALUES.F_NAN,0.,10.,100.])
  0   0   0   0
% Program caused arithmetic error: Floating illegal operand
IDL> print,bytscl([!VALUES.F_NAN,0.,10.,100.],/NAN)
  0   0  25 255
```

What it's good for is ensuring that the rest of your finite values are
scaled appropriately. The problem, of course, is that:

```
IDL> print,max([!VALUES.F_NAN,0.,10.,100.])
        NaN
```

so, absent the /NAN, this becomes the maximum value to scale the array

to, with the obvious deleterious side effect that all scaled values are now NaN (think of NaN like a virus, infecting everything it touches). Since there's no byte value representing NaN, as there is a float value, they had to pick something, so they picked 0b, which seems as good as anything else. Any value you pick would be degenerate with real data.

If you'd like to separate out the NaN's, I'd use:

b=bytscl(a,/NAN,TOP=254)+finite(a)

Now 0 is definitely a NaN, and 1-255 is real data. Yes, there is a duplicate check for NaN's implicit here. If you rescale the image many times without changing the data, you might cache finite(a) for a bit of speedup. I even go so far as to check if there are any non-finite values and use:

b=bytscl(a,NAN=self.non_finite,TOP=254)+self.non_finite?self .finite:1

The reason? Adding /NaN slows most functions down by about a factor of 2.

JD
