Subject: Re: Fanning's LogScl routine + Colorbar??
Posted by pgrigis on Mon, 26 Jul 2010 14:59:43 GMT
View Forum Message <> Reply to Message

On Jul 24, 11:40 am, David Fanning <n...@dfanning.com> wrote:
> Joe Daal writes:
>> I am using the logscl to enhance the contrast of an image, something
>> like:
>
>> loadct,39
>> image = alog10(lin_image)
>> imdisp, logscl(image, min=min(image), max=max(image), Exponent=8,
>> Mean=0.65)
>
>> where image values vary from -1.34 to +2.05, with zeroes included.
>> The image looks nice for what I want, but how do I reflect a correct
>> colorbar, either for real values or the scaled ones? What is
>> logarithmic, the ticks or the colors?
>
> After thinking about this some more this morning, I decided
> I would write an article about it. Normally, when I write an
> article I am about 90% sure I know what I'm talking about.
> (My wife says I have been overly optimistic my whole life!)
>
> In this case, it's more like 50%. But I figure the worst
> thing that could happen to me would be that I might learn
> something. ;-)
>
> You can find the article here:
>
> http://www.dfanning.com/ip_tips/logscaledbar.html
>

Well, let me try to explain my argument.

Basically, there are 2 (mutually exclusive) ways to proceed
with this:
a) rescaling the color tables
b) rescaling the data

These 2 ways are *not* mathematically equivalent, as explained below.
In general a) will lead to the rescaled image having (in some pixels)
colors that were not present in the image before rescaling, while b)
will not. Think of a) as being a more powerful transformation - but
with
great power come great responsibility as you all know :)

Personally I dislike a) creating new colors not present in the

original
image and therefore I stick to b).

Here the mathematical argument:
what we think of a "color table" is the combination of 2 operations.

The first is the process of assigning every pixel of the image an
index between
0 and N-1 (N can be any number, 256 is often used but it's important
to
realize that this number is not tied to that). You can think of this
as
a function f going from the real numbers to [0,1,...,N-1].

The second is the process that assigns every index a color (in the
case
of the current hardware, a color is a triple of bytes R, G, B). This
is a set of three functions R,G,B going from [0,1,...,N-1] to
[0,1,...,255].
The fact that we have 256 shades for 3 main colors is fixed and
limited
by the hardware.

To display an image "im" we have to compute R(f(im)),G(f(im)),B(f(im))
for
all pixels - this is what we mean by using a color table.

Now in case we are not happy with the result we can try rescaling
using the a) or b) process.

The a) rescaling means we have functions R2,G2,B2 that go from
[0,1,...,255]
to [0,1,...,255]. We then display the image R2(R(f(im))),
G2(G(f(im))), B2(B(f(im))).
Depending on the details of R2,G2,B2 it's quite easy to create new
colors by this
transformation (why? because there are only N different triples before
the
transformation, and 16777216 (!) different triples that they can be
transformed
into).

The b) rescaling means we have a function h that goes from
[0,1,...,N-1] to
[0,1,...,N-1]. We then display the image
R(h(f(im)),G(h(f(im)),B(h(f(im))).
Because the R,G,B functions themselves are not changed, the new image

can only
consist of colors in the color table (i.e. no new colors will appear).

That sums it up... Hopefully this helps shed some light (or muddle
up the water even more instead).

Ciao,
Paolo

> 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.")
