Subject: New Image Processing Routines
Posted by David Fanning on Wed, 22 Feb 2006 20:47:21 GMT
View Forum Message <> Reply to Message

Folks,

I've been boning up on astronomy lately (Stars and their Spectra by James Kaler just arrived in today's mail) because I want to build an astronomy image viewer. I've had some tools to work with images, but they haven't worked terribly well with the perverse images generated by astronomers and saved in FITS files. At least, that has been my experience with them. Maybe I only get sent the worst of the lot, I don't know.

And I don't know if it is my imagination or whether FITS images are just FULL of values close to zero, but I generate so many &%$@# floating underflow messages when I deal with them that it just depresses me. Frankly, if I never see another floating underflow message in my life it will be too soon.

All this prelude to saying that I spent the past couple of days with my nose in that extraordinary book, Digital Image Processing, 2nd Edition, by Gonzalez and Woods, and it's even more useful (to me, anyway) companion, Digital Image Processing with Matlab. Damn, these are two good books!

I got excited about Chapter 3, Intensity Transformation Functions, and decided to replicate the programs described in that chapter in IDL. Intensity transformations allow you to manipulate contrast in images, which is important not only in astronomy, but in many other image processing fields. The programs below, for example, have already been helpful to me in preprocessing medical images prior to snaking with active contours.

Here are brief descriptions of the new programs I've written and downloaded to my web page:

http://www.dfanning.com/documents/programs.html

IMGSCL -- I think of this as BYTSCL on steroids. It scales your data into 0 to 255, but you can choose a power law (gamma) scaling in addition to a linear scaling. Moreover, you can choose the output values directly, and you can get a negative image as well as the normal positive image just by setting the Negative keyword.

LOGSCL -- Similar to IMGSCL, except instead of a
power-law log transformation, it uses a log transformation.
The advantage of this is that you can compress values at either
end of a data range, and stretch the data centered about
a specified value, called the MEAN in the program.

XSTRETCH -- This old program has been completely rewritten to
serve as a graphical user interface to IMGSCL. It now works
perfectly with all the gnarly FITS and DICOM images I have laying
around here, in addition to the old standards in the IDL examples/data
directory. If contrast is your problem, I feel confident saying
you can use this to solve it. The program displays the image
histogram as an aid to selecting the proper contrast stretch.

I have also improved the way this program works with your
own programs or objects. So, it is possible to just use the
control panel and have the image displayed when and where
you like.

I have also beefed up SCALE_VECTOR, a program that I rely
on more and more to do all sorts of data scaling tasks, and
I have added a routine called CONVERT_TO_TYPE which can
convert its input to various data types at run-time. All
of these programs now fix the problems that cause data
underflow, or turn the damn messages OFF when I can't do
a damn thing about it. (For example, just taking the MIN
or MAX of an array containing values close to zero cause
these warnings.)

Even as I am writing this note, I can think of new things
to do with some of these programs, so check back often.

And, as always, I'm interested in errors you might find.
(I found another as I was downloading the "final" version. :-)

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: New Image Processing Routines
Posted by David Fanning on Fri, 21 Apr 2006 13:45:27 GMT

Marshall Perrin writes:

> But on to my main point. I think your ASINHSCL can be simplified considerably,
> because the Alpha parameter is unnecessary. Or rather, Alpha and Beta aren't
> independent. What actually sets the shape of the scaling law is their product.
>
> Try varying them inversely (i.e. move Alpha up by a factor of 5, Beta
> down by a factor of 5, so that their product remains constant) and I
> think you'll see that the scaled image is unchanged. I've also
> verified this behavior with some simple 1D plots (code attached
> below). I've thought about the equations for a bit, but I don't yet
> have a rigorous proof that this is true, but it is in all the
> numerical examples I've tried. Then again, it's 4:30 am and
> even on astronomer hours that's a bad time for doing math, so maybe
> I'll figure it out tomorrow.
>
> In any case, I think you can just remove alpha entirely and use beta
> alone to tune the scaling. I note that Lupton et al. only mention
> Beta as a tuning parameter as far as I can tell. I see there's
> an alpha in Eric Sheldon's tvasinh.pro that you based your code on,
> but I have no idea where he got it from...

Yes, I discovered this disquieting state of affairs myself.
But after playing with it late into the night, I had pretty
much convinced myself that if it didn't exactly do any good
to have two parameters, it didn't appear to do any harm, either.
And I noticed if I held my mouth just right and squinted,
I could even justify the two parameters as doing something
different, which made the shape of the curve easier to
control. (I don't remember at the moment, but I thought one
factor affected the steepness of the curve, while the other
affected where the "bend" in the curve started.)

I've since moved on to more pressing things, but maybe
I'll have to revisit this. Let me know if a few hours of
sleep leads you to some more ideas. :-)

Cheers,

David

Cheers,

David
--
David Fanning, Ph.D.

Subject: Re: New Image Processing Routines
Posted by Marshall Perrin on Fri, 21 Apr 2006 17:24:01 GMT
View Forum Message <> Reply to Message

David Fanning <davidf@dfanning.com> wrote:
> I've since moved on to more pressing things, but maybe
> I'll have to revisit this. Let me know if a few hours of
> sleep leads you to some more ideas. :-)

Actually, it was while brushing my teeth that I had the epiphany.
The relelvant line of code is the scaling, of course:

```
 output =  Scale_Vector(ASinhScl_ASinh(alpha*beta*Temporary(output))/be ta, $
     minOut, maxOut, /NAN, Double=1)
```

The only way in which alpha affects things is as a multiplicative constant
times the input (also named 'output' above just to be tricky :-) We know that
it's the ratio of beta to the input that sets the amount of nonlinearity
in the data. Quoting from Lupton et al 1999, right after eq. 4:

> For x -> \infinity, \mu approaches m for any choice of \beta. On
> the other hand, when |x| <~ b, \mu is linear in x.

So it's the ratio of beta to x that matters, and since alpha just has the
effect of scaling x, it doesn't actually add an additional degree of freedom.
I'm fully convinced now that alpha doesn't give you anything new.

Next problem: the beta in your code (and in all the other IDL implementations
of asinh scaling, as far as I can tell!) is actually the *inverse* of the
parameter b in Lupton et al. Quoting now from Lupton et al. 2004, shortly
after Eq. 2:

> We take F = arcsinh(x/Beta), where the softening parameter Beta is
> chosen to bring out the desired details.

So they say they are dividing the input by Beta, while you're multiplying it.
Actually, they *say* they are dividing the input by Beta, but if you download
their code from http://cosmo.nyu.edu/hogg/visualization/ and look at
nw_asinh_fit.pro, the relevant line of code is

```
   val = asinh(radius*nonlinearity)/nonlinearity
```

The same approach is used in Dave Schlegel's djs_rgb_make available in his

IDLUTILS library. So it seems that they've defined nonlinearity = 1./beta . Right? And thus your Beta is the inverse of theirs.

Does this actually matter? No, not really. Either way works fine. It's just a slightly confusing state of affairs for anyone trying to match up IDL code to published algorithms. I *think* the simplest solution is to rename in your code "beta" to "nonlinearity" and put in a note that nonlinearity = 1./b. But hey, who am I to tell you what to name your variables? :-)

Another thing possibly worth adding to the documentation is that if you have some estimate of the noise in your image, then setting nonlinearity = 1./noiselevel (or equivalently, beta=noiselevel) does a decent job at showing the full dynamic range of the image from the peak down to the read noise. Actually I've found that some small multiple of the noiselevel works best, say 5. That makes the asinh scaled image linear in the range of 0 - 5 sigma, then logarithmic for everything brighter.

Oh, this is so slick. I've *got* to start using this for all my plots.

- Marshall

Subject: Re: New Image Processing Routines
Posted by Marshall Perrin on Fri, 21 Apr 2006 19:31:08 GMT
View Forum Message <> Reply to Message

Marshall Perrin <mperrin+news@cymric.berkeley.edu> wrote:
> Another thing possibly worth adding to the documentation is that if
> you have some estimate of the noise in your image, then setting
> nonlinearity = 1./noiselevel (or equivalently, beta=noiselevel) does
> a decent job at showing the full dynamic range of the image from the
> peak down to the read noise.

Never mind, I take the above back. It was only true because I happened to be using an input data set whose range was roughly 0-1. For inputs with other ranges, the above doesn't work out, because the ratio between beta and the input is changed by the first call to scale_vector. Still, the basic idea is right, there's just a few constants to keep track of. I'll have to play with this a bit more.

- Marshall

Subject: Re: New Image Processing Routines
Posted by David Fanning on Mon, 24 Apr 2006 19:24:54 GMT

Marshall Perrin writes:

> Actually, it was while brushing my teeth that I had the epiphany.
> The relelvant line of code is the scaling, of course:
>
> output = Scale_Vector(ASinhScl_ASinh(alpha*beta*Temporary(output))/be ta, $
> minOut, maxOut, /NAN, Double=1)
>
> The only way in which alpha affects things is as a multiplicative constant
> times the input (also named 'output' above just to be tricky :-) We know that
> it's the ratio of beta to the input that sets the amount of nonlinearity
> in the data. Quoting from Lupton et al 1999, right after eq. 4:
>
> For x -> \infinity, \mu approaches m for any choice of \beta. On
> the other hand, when |x| <~ b, \mu is linear in x.
>
> So it's the ratio of beta to x that matters, and since alpha just has the
> effect of scaling x, it doesn't actually add an additional degree of freedom.
> I'm fully convinced now that alpha doesn't give you anything new.

Ok, I've convinced myself I agree with you. :-)

> Next problem: the beta in your code (and in all the other IDL implementations
> of asinh scaling, as far as I can tell!) is actually the *inverse* of the
> parameter b in Lupton et al. Quoting now from Lupton et al. 2004, shortly
> after Eq. 2:
>
> We take F = arcsinh(x/Beta), where the softening parameter Beta is
> chosen to bring out the desired details.
>
> So they say they are dividing the input by Beta, while you're multiplying it.
> Actually, they *say* they are dividing the input by Beta, but if you download
> their code from http://cosmo.nyu.edu/hogg/visualization/ and look at
> nw_asinh_fit.pro, the relevant line of code is
>
> val = asinh(radius*nonlinearity)/nonlinearity
>
> The same approach is used in Dave Schlegel's djs_rgb_make available in his
> IDLUTILS library. So it seems that they've defined nonlinearity = 1./beta .
> Right? And thus your Beta is the inverse of theirs.
>
> Does this actually matter? No, not really. Either way works fine. It's
> just a slightly confusing state of affairs for anyone trying to match
> up IDL code to published algorithms. I *think* the simplest solution
> is to rename in your code "beta" to "nonlinearity" and put in a note
> that nonlinearity = 1./b. But hey, who am I to tell you what to name
> your variables? :-)

I'm still confused about this part. I've named the variable nonlinearity, and I am happy that a value of 0 means a linear fit. Increasing the value of the variable changes the shape of the curve, so that the linear part gets shorter and steeper and the logarithmic part gets longer and flatter with increasing values. But, the values have to increase in a logarithmic fashion for this to make much sense. This suggests to me that I haven't hit on exactly the right formula yet.

In any case, I modified XSTRETCH to take advantage of the new ASINHSCL routine. And while I was at it, added several more features. You can now specify the gamma value, as well as the nonlinearity value with a combobox widget. This allows you to select decent choices off a pull-down menu, but if you don't like the choices I give you, you can just type your own values in. I even put in the linear "scaling curve" just for JD. (I previously had assumed people would know what a linear scaling curve would look like. But JD's probably right. You never know what people need. :-)

As a working hypothesis I think my "nonlinearity" parameter doesn't directly correlate with Lupton's "beta" parameter, but I think this is because I scale the image data into the range of 0 to 1 before I apply the scaling. This is necessary, in my case, because I can't rely on image data values falling into a particular range. I need to start with something I can depend on.

The result is certainly more intuitive, if not more correct. :-)

http://www.dfanning.com/programs/asinhscl.pro
http://www.dfanning.com/programs/xstretch.pro

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: New Image Processing Routines
Posted by Marshall Perrin on Tue, 25 Apr 2006 04:47:51 GMT
View Forum Message <> Reply to Message

David Fanning <davidf@dfanning.com> wrote:
> As a working hypothesis I think my "nonlinearity" parameter
> doesn't directly correlate with Lupton's "beta" parameter,
> but I think this is because I scale the image data into
> the range of 0 to 1 before I apply the scaling. This is
> necessary, in my case, because I can't rely on image data
> values falling into a particular range. I need to start
> with something I can depend on.

I think that you're correct here, but the "right" correlation between nonlinearity and beta isn't too hard. To convert Lupton's beta into nonlinearity, we just have to account for what Scale_Vector is doing, right?

```
scaled_beta = (beta - MinValue) / (MaxValue - MinValue)
nonlinearity = 1./scaled_beta
```

This is actually a useful way to parametrize things, because then you can let beta equal whatever you think the noise level in the image is, and the above code will compute the nonlinearity factor which lets you simultaneously see the noise *and* the image maximum. So I don't know if beta is actually -entirely- obsolete just yet. Alpha, now, that's another story. :-)

- Marshall

Subject: Re: New Image Processing Routines
Posted by Marshall Perrin on Tue, 25 Apr 2006 06:48:59 GMT
View Forum Message <> Reply to Message

OK, while we're talking about this, I've got another gripe, I mean, constructive comment about asinhscl. This time, it's about reproducibility of scaling. I just spent an hour tracking down some strange behavior with some plots, which I ultimately traced to how Scale_Vector works.

The scenario is this. I have a couple of images which I want to display with the same scaling, and then I also want to display a color bar. I want to ensure that all the images have precisely the same scaling, so I say

```
minval = min(image1) < min(image2) < min(image3)
maxval = min(image1) > min(image2) > min(image3)
```

and then I can call

```
tvimage,asinhscl(image1,min=minval,max=maxval)
tvimage,asinhscl(image2,min=minval,max=maxval)
tvimage,asinhscl(image3,min=minval,max=maxval)
```

However, this did *NOT* do what I expected. Pixels in the images which should
have had the exact same output greyscale ended up having slightly different
values.

I'll cut to the chase. The problem is that some images have maxes and mins
different from the overall max and min, so the first Scale_Image call in
asinhscl maps them into, say 0.03-0.98 instead of the full range 0.0-1.0. Then
the *second* call to Scale_Vector doesn't have explicit min and max set, so
it maps asinh(0.03) to 0 and asinh(0.98) to 1. Thus the color map
is slightly different for each image. Whoops!

The fix is relatively simple - figure out where asinh sends 0 and 1, and
then set those as the range for the second Scale_Vector:

```
 extrema = asinhscl_asinh([0.0,1.0]*nonlinearity)
 output =  Scale_Vector(ASinhScl_ASinh(Temporary(output)*nonlinearity), $
    minOut, maxOut, MinValue=extrema[0], MaxValue=extrema[1], /NAN, Double=1)
```

I think at this point we're starting to converge on a final version.
I've also got a hacked-up version of your colorbar.pro for making
asinh scaled colorbars, but I want to polish it a bit more before releasing
it into the wild. :-)

 - Marshall

Subject: Re: New Image Processing Routines
Posted by David Fanning on Tue, 25 Apr 2006 13:19:09 GMT
View Forum Message <> Reply to Message

Marshall Perrin writes:

> The scenario is this. I have a couple of images which I want to display
> with the same scaling, and then I also want to display a color bar.
> I want to ensure that all the images have precisely the same scaling,
> so I say
> minval = min(image1) < min(image2) < min(image3)
> maxval = min(image1) > min(image2) > min(image3)
>
> and then I can call
> tvimage,asinhscl(image1,min=minval,max=maxval)
> tvimage,asinhscl(image2,min=minval,max=maxval)
> tvimage,asinhscl(image3,min=minval,max=maxval)
>
> However, this did *NOT* do what I expected. Pixels in the images which should
> have had the exact same output greyscale ended up having slightly different
> values.

Sigh... I had something else planned for this morning. :-(

Let me ask you a question, though, before I get started.
In this scenario, do the original images have starting values
between 0 and 1? Driving back from tennis last night I
had a sudden dark thought that this could be the source of
a problem I thought I saw the other day. This message
casts an ominous pall on an already gloomy day here in Colorado.

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: New Image Processing Routines
Posted by David Fanning on Tue, 25 Apr 2006 16:01:44 GMT
View Forum Message <> Reply to Message

Marshall Perrin writes:

> I think that you're correct here, but the "right" correlation between
> nonlinearity and beta isn't too hard. To convert Lupton's beta into
> nonlinearity, we just have to account for what Scale_Vector is doing,
> right?
>
> scaled_beta = (beta - MinValue) / (MaxValue - MinValue)
> nonlinearity = 1./scaled_beta
>
> This is actually a useful way to parametrize things, because then
> you can let beta equal whatever you think the noise level in the
> image is, and the above code will compute the nonlinearity factor
> which lets you simultaneously see the noise *and* the image maximum.

OK, new versions of both ASINHSCL and XSTRETCH up on my
web page. I've removed the NONLINEARITY keyword (which my
fingers could never type correctly anyway) and replaced
it with a BETA keyword, which directly corresponds now
(I'm pretty sure) with Lupton's "softening parameter".
In any case, I have a rational explanation for their
default value of 3, which I never had before. That seems
like progress to me. :-)

I kept the nonlinearity variable internally, because it
gives me a good sense of what is actually happening to

the equation, but I create it with your formula above.

I also fixed the scaling problems you were having with
ASINHSCL and the COLORBAR program. (Although I am not
fully finished with the testing yet.)

Let me know. I really appreciate the help with this! :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: New Image Processing Routines
Posted by Marshall Perrin on Tue, 25 Apr 2006 16:38:48 GMT
View Forum Message <> Reply to Message

David Fanning <davidf@dfanning.com> wrote:
> Sigh... I had something else planned for this morning. :-(

The best-laid plans of mice and IDL programmers... If it makes you
feel any better, I was supposed to be doing real work on my thesis
these last few days, instead of just figuring out how to scale pretty
pictures for my thesis. :-)

> Let me ask you a question, though, before I get started.
> In this scenario, do the original images have starting values
> between 0 and 1? Driving back from tennis last night I
> had a sudden dark thought that this could be the source of
> a problem I thought I saw the other day. This message
> casts an ominous pall on an already gloomy day here in Colorado.

In this case, yes, most of the pixels in the original images were
between 0 and 0.5, not for any fundamental reason but because that's where
the values happened to fall in the units I was working in. There were a lot
of slightly-negative pixels too, from read noise, so the total image range
was something like -0.01 to 0.5.

- Marshall
