IMAGE DISPLAY, ANNOTATION, AND EXAMINATION WITH TIM ROBISHAW'S `display.pro` AND RELATED PROCEDURES
April 2, 2009

Carl Heiles, Tim Robishaw

Contents

1 INTRODUCTION **2**

2 USING TIM ROBISHAW'S `display` TO MAKE AND EXAMINE 1D OR 3D IMAGES **3**

2.1 1d Images 3

2.2 3d Images 4

2.3 Examine the 1d Image on Your Terminal 4

2.3.1 Interactively Locate Points Using `trc` 4

2.3.2 Interactively Plot 1d Image Intensities Versus x or y Using `trp` 5

3 ADD A COLORBAR! USING TIM ROBISHAW'S `colorbar` **5**

3.1 A 1d Colorbar 6

3.2 A 2d Colorbar 6

4 IF YOU WANT AN IMAGE AND COLORBAR ON POSTSCRIPT... **7**

5 USING `dsply2d` FOR 2D IMAGES **8**

6 USING `rgbimg` FOR 3D COLOR IMAGES **9**

1. INTRODUCTION

Tim Robishaw's `display.pro` does most of what we cover in other handouts[1], but much more simply. Here's an introduction to this and related programs. We emphasize that this is an *introduction* to those programs; for more details, look at their documentation (in IDL, that's `doc, 'display'` for `display.pro`). One more thing: while `display.pro` does things more simply, those handouts describe how to things more flexibly (e.g., sizing and placement of display windows and colorbars)—and less simply.

Important! There are various authors who wrote different versions of `display` and `colorbar`; we treat, and recommend, *Robishaw's* versions. The Robishaw procedures we cover here include:

1. `display.pro` . This displays 1d or 3d images, draws axes, labels axes, properly bytescales the images (either automatically or according to user specs), changes contrast...
2. `trc.pro` . Shorthand for trcursor, this slaps a nice crosshairs on the screen and, when you push the mouse button, records the image value and the x/y positions. `trc` works for ordinary graphs, too, for which it is very handy!
3. `trp.pro` . Shorthand for trprofiles, this plots image profiles (the image intensity versus the x or y coordinate).
4. `colorbar.pro` . Puts a colorbar near your displayed image.
5. *Important!* If you are making a postscript file output, then see §4 below!

In short, the first three of these Robishaw procedures do what the native IDL procedures `tv`, `rdpix`, and `profiles` do, except that they can give the x/y values in designated coordinate values instead of pixels, do a few other nice things, and work better.

Let's get started at the same place as in §3 of IDIDL. Specifically, we'll start with 1-d images, like a greyscale or false-color image, which means you are using a 256-entry color table, i.e. you have turned color decomposition off[2]. (If this is Greek to you, read the first two sections of IDIDL.)

---

[1]Particularly the three handouts "IDIDL—Image Display and Manipulation in IDL" (ididl.ps) "1d2d3d: One, Two, and Three Dimensional Color Images" (newdimensionalcolor.ps). "IACIDL: Image Annotation and Colorbars in IDL" (annotated_images.ps).

[2]You do this by typing `device, decomposed=0` .

2. USING TIM ROBISHAW'S `display` TO MAKE AND EXAMINE 1D OR 3D IMAGES

Here we assume you are writing to the terminal window, not postscript; see §4 for postscript. Before starting, set `ps=0`; this tells the `display` and `colorbar` programs what you're doing.

2.1. 1d Images

First, generate an image[3]. In IDIDL, we matched the window size to the image size. When using `display`, you don't need to do that; it's done automatically, and it will even generate a window for you if you don't have one open. If you want a larger image on your screen, you can define a window of your desired size and `display` will fill it. You have to be careful, though, about the effects of reducing the size (number of pixels) of an image because the interpolation process can lose information, depending on how it's done. The discussion of `congrid` in IDIDL explains these concerns, which apply to `display` even though it doesn't use `congrid`.

A good way to use `display` for this particular image:

```
x=findgen(480)* (360./480.)
y=findgen(240)* (180./240.) - 90.
ps=0

display, img, x, y, min=0, max=2000, out=out, ps=ps, $
xtit='Galactic Longitude', ytit='Galactic latitude', title= 'C-Band ROSAT Sky'
```

Some explanation:

1. `x` and `y` are the axes values. The angles run from 0 to 360 in `x` and from –90 to 90 in `y`. The first dimension of `img` must have the same number of elements as `x` while the second dimension of `img` must have the same number of elements as `y`.

[3]There's a nice 1d image of the X-ray sky, obtained by the ROSAT satellite, in `http://astro.berkeley.edu/~heiles/handouts/rass_c.fits` . Copy this file to the disk area where you are running IDL. To accomplish this, type `wget http://astro.berkeley.edu/~heiles/handouts/rass_c.fits` . This file is in a format called "FITS" format, which is the standard format of most astronomical images. To read the data file into an array called `img`, it is easiest to use the IDL procedure called "readfits", which resides in the Goddard IDL library, which, in turn, is already in your IDL path. All you have to do is type `img = readfits('rass_c.fits', headerinfo)` . This returns two arrays: the image array (`img`) and information about the image (`headerinfo`); type `print, headerinfo` to see the header information. Now type `help, img` and IDL will tell you that it is a 480 × 240 FLOAT array.

2. `min` and `max` say that the minimum data value will be converted to 0 (the minimum intensity, black for a greyscale image) and the maximum to 255 (white).

3. If you wanted to make a false-color image instead of greyscale, load in your desired color table. For example, if you want to use a native IDL color table, e.g. number 5, then before calling `display` type `loadct, 5`.

4. The x, y, and image titles speak for themselves.

5. `out` is an output structure that contains information used for examining and manipulating the image; check it out. It is used directly as an input to `trp` .

In addition, there are some useful additional keywords or optional inputs for `display`. Some of the more useful include:

1. `aspect` allows you to change the shape of the image. `aspect` is the ratio of the x to the y dimensions of the output image.

2. `negative`: if you set this keyword, then the color table is reversed. That is, instead of black/white being min/max, black/white is max/min. For greyscale images to be printed on paper, or projected in a powerpoint presentation, you *almost always* want to set `negative`!

3. If you want to change the contrast of the image, you must do it yourself before calling display. For example, to increase the contrast so that weak features are highlighted, one often raises the image values to a power less than one. This power is usually called `gamma`.

4. If you want to write the image to a postscript file, see §4.

2.2. 3d Images

Everything here is exactly as above, except that **(1)** your image must be a truecolor image of dimension (3, nx, ny) [i.e., 3 colors by (nx/ny) (horizontal/vertical) pixels]; and **(2)** you need to set the `true` keyword when calling `display`.

2.3. Examine the 1d Image on Your Terminal

2.3.1. Interactively Locate Points Using `trc`

`trc` is just a wrapper that calls `tr_rdplot` with useful settings. Type

```
trc, x, y
```

Then move the cursor inside the plot window. Left clicks write and record the `(x,y)` values in the coordinate system you select (i.e., `data, device, normal`; default is `data`). A middle or left click will exit the program. Notice that the crosshairs are clipped at the axes: it's often useful to make the crosshairs extend to the edges of the window. This is accomplished by passing the `/noclip` keyword, and since we can use minimum matching in IDL, `/noc` will be good enough:

```
trc, x, y, /noc
```

If you don't want to store the positions of your left-clicks in `x` and `y`, then you don't need to include them in the procedure call. For more details, see the documentation of `trc` and `tr_rdplot`.

2.3.2. Interactively Plot 1d Image Intensities Versus x or y Using `trp`

`trp` is just a wrapper that calls `tr_profiles` with useful settings. Type

```
trp, out
```

Notice that a new window has been created. Move the cursor inside the plot window containing the displayed image. You now see the profiles in the newly created window (labeled *Profiles*). Initially, you see intensity versus the `x` data coordinates. Left click gives intensity versus the `y` data coordinates. Middle click prints the x/y values. Right click exits the program and closes the profiles window. For more, see the documentation for `trp` and `tr_profiles`.

Normally `trp` plots profiles of the original, unscaled image. However, if you have scaled the displayed image using `min` and/or `max`, and if you want to get profiles of this scaled image, type

```
trp, out, /scaled
```

Obviously, in order for all this to work, you need to have used the `OUT` keyword for `display` prior to calling `trp`.

3. ADD A COLORBAR! USING TIM ROBISHAW'S `colorbar`

Displaying an image without a colorbar is like displaying a graph with axes that have no numbers or labels. It's easy to add a colorbar with Tim's `colorbar`. This procedure does either 1d or 2d colorbars; see the images in in the handout "1d2d3d: One, Two, and Three Dimensional Color Images". Here we first describe what to do for a 1d colorbar; next we describe the 2d case.

In either case, the first job is to decide where to put the colorbaar. Decide if you want a horizontal colorbar (usually on the top) or a vertical one (usually on the right hand side). There's a keyword `vertical`; set it appropriately.

You need enough space for the colorbar. To make enough space on the terminal, use a bigger

x-window; to make enough on the postscript window, see §4.

Then define `position`, a 4-element vector that specifies the normalized coordinates[4] of the corners of the colorbar. The 4 elements are in order (lowleftx, lowlefty, uprgtx, uprgty). For example, for a horizontal colorbar on the top you might choose something like `position = [ 0.1, 0.8, 0.9, 0.9]` .

The remaining jobs depend on whether you are making a 1d or 2d colorbar.

3.1. A 1d Colorbar

1. Define the data values that correspond to the min and max of the image brightness; for the description of `display` above, these would be `min` and `max`. Set `crange=[min, max]`.

2. If you raised the data to a power (`gamma`) to change the contrast, then set `cgamma = gamma`. If not, set `cgamma=1` or leave it undefined.

3. Define the name of the displayed quantity, e.g. `xtitle='Xray intensity'`.

4. Then make the colorbar:

```
colorbar, POSITION=position, ps=ps, CRANGE=crange, CGAMMA=cgamma, $
        NEGATIVE=negative, VERTICAL=vertical, XTITLE=xtitle
```

   For more info on colorbar parameters, see its documentation.

3.2. A 2d Colorbar

1. For a 2d image you need to specify the min and max for both the color image (`cmin, cmax`) and the intensity image (`imin, imax`). These min, max values are specified as 2-element vectors. So for the color image, it's `crange= [cmin, cmax]` and for the intensity image it's `irange= [imin, imax]`. Similarly, each image has its own gamma (the power to which the data numbers are raised): `cgamma` and `igamma`. It should be obvious that these gamma values should correspond to what you used when you displayed the image. The titles on the colorbar are `xtitle` (the name of the variable that represents color, e.g. velocity) and `ytitle`.

2. Finally, for 2d color image and colorbar you are using a 256-entry colortable to define the colors. The 256 (r,g,b) intensities are represented with a 256×3 array called `colr`. If you use the physiologically-correct colortable discussed in the handout "1d2d3d: One, Two, and Three Dimensional Color Images", then you get this array from the procedure `pseudo_ch,`

[4]Normalized coordinates range from 0 to 1 and are the fraction of the size of the window.

`colr`. We assume this in our example just below; if you get `colr` in another way, you still need to `loadct,0` before calling `colorbar`.

3. Finally, invoke the colorbar procedure:

```
pseudo_ch, colr
loadct,0
colorbar, POSITION=position, RGB=colr, ps=ps, $
CRANGE=crange, IRANGE=irange, CGAMMA=cgamma, IGAMMA=igamma, $
        NEGATIVE=negative, VERTICAL=vertical, $
        XTITLE=xtitle, YTITLE=ytitle
```

4. IF YOU WANT AN IMAGE AND COLORBAR ON POSTSCRIPT...

If you want a nice image/colorbar in postscript, then do the following (We're repeating the individual contents of §2.1 and §3.1, just so you see how easy it is)[5]:

1. You must tell `display` and `colorbar` that you're writing to postscript, so

   `ps=1`

2. Open the postscript device, making sure to specify the size; the size must be big enough to accommodate both the image and the colorbar. For example:

   ```
   psopen, 'xray.ps', /times, /bold, /isolatin1, /color, xsize=8, ysize=8, /inch
   ```

   which writes to the file `xray.ps`, uses the Times/Bold font, allows color in the image, and sets the x and y sizes to 8 and 8 inches.

3. Display the image using `display`, setting ps. You need to specify the size of the image and, in particular, you need to make it smaller than the postscript window size; this leaves room for the colorbar. So you might say

```
x=findgen(480)* (360./480.)
y=findgen(240)* (180./240.) - 90.
xsize=8
ysize=5
min=0
max=2000
display, img, x, y, ps=ps, min=min, max=max, out=out, $
```

[5] If you don't want it on postscript, don't do the first two steps—i.e., set `ps=0` and don't invoke `psopen` .

```
xsize=xsize, ysize=ysize, /inch, $
xtit='Galactic Longitude', ytit='Galactic latitude', title= 'C-Band ROSAT Sky'
```

Notes: **(1)** With a ps window ysize of 8 inches and an image ysize of 5 inches, you have 3 vertical inches left for the colorbar. **(2)** If you want to annotate the image, do so here using `xyouts` .

4. Now we display the colorbar, as above, setting ps. Manipulate the `position` coordinates, using trial and error, to make it look visually good. And if we're in postscript, we close the file and properly return to x windows.

```
position = [0.15, 0.79, 0.85, 0.88]
xtitle=  'ROSAT counts'
colorbar, POSITION=position, ps=ps, CRANGE=crange, CGAMMA=cgamma, $
        NEGATIVE=negative, VERTICAL=vertical, XTITLE=xtitle
if ps then psclose
if ps then setcolors, /sys
```

5. USING `dsply2d` FOR 2D IMAGES

This poorly-documented procedure is quick and automatically gives you a nice colorbar. However, it requires your images to be 541×541 pixels. If they are not this size, the procedure resizes them using `congrid`; the image always comes out square. A more flexible alternative is the 2d example in the handout "1d2d3d: One, Two, and Three Dimensional Color Images".

2d images synthesize color and intensity, so these plots explicitly require information for the two dimensions—i.e., the two images. Define

1. `c_img` and `i_img` as the color image and intensity image, respectively.
2. `xaxis, yaxis` as the (x,y) pixel values (like `x,y` for `display`).
3. `c_title` and `y_title` as the titles for the two images
4. Keyword `c_range`: a 2-element vector specifying the max and min of color used for labeling (e.g., if colors correspond to a velocity range of -10 to 10 km/s, then `c_range = [-10,10]`).
5. Keywords `intmin, intmax` as the minimum/maximum intensity values for the intensity images (like `min,max` for `display`).
6. Keyword `gamma` as the contrast parameter (the intensity image is raised to this power).
7. Keywords `xaxlbl, yaxlbl` as the x- and y-axis labels.

8. For postscript, set keywords `ps`, `filename` and, if you want encapsulated postscript, `encapsulated`.

There are some other keywords; you can look at the code and experiment. As I said above, for more flexibility look at the 2d example in the handout "1d2d3d: One, Two, and Three Dimensional Color Images".

6. USING `rgbimg` FOR 3D COLOR IMAGES

We have a clumsily-written but reliable procedure to produce a 3d image with a fancy colorbar like those of Figure 6 in the handout "1d2d3d: One, Two, and Three Dimensional Color Images". A severe restriction is that the image must have dimensions 541×541; you can use `congrid` to convert your image to this size (but see the cautions about `congrid` in in the handout "1d2d3d: One, Two, and Three Dimensional Color Images"). The procedure is called `rgbimg.pro` and it is reasonably well-documented.

To see what kind of image and colorbar this produces, see the handout "1d2d3d: One, Two, and Three Dimensional Color Images".