Lab 1: Taste the Rainbow

Colors and Visible Images (30 points possible)

How to submit this assignment

A PDF of the report for this lab must be uploaded to Gradescope by 10 am on Thursday, January 27 2022 (check Brightspace for updates on due date).

Gradescope sign up instructions:

(1) Go to gradescope.com
(2) Click Sign Up → Sign up as student
(3) Enter these fields:
    a. Entry code: **KY458G**
    b. Your full name
    c. Purdue email address (your actual career account address please)
    d. Purdue ID
(4) Choose a password and confirm your email as instructed
(5) Next time you log in, you should see EAPS 577 listed on your dashboard.

Lab report submission instructions:

(1) Complete the report, including pasting all requested images/plots/etc.
(2) Reduce the file size of the report (File -> Reduce File Size).
(3) Save your completed Word document as a PDF (File -> Save As -> File Format: PDF).
(4) Log into Gradescope using your email address and password.
(5) Go to EAPS 577 on your Dashboard.
(6) Click on Lab 1 Report.
(7) Click on PDF.
(8) Open your saved PDF of the report.
(9) Your last step is to tell us which page corresponds to each question on the assignment. You will see a list of all the questions, and images of the pages of your pdf. For each question, click the question and then click the page(s) that contains your answer to that question. You can use the SHIFT key to select multiple questions at a time and assign them to the same page.
(10) Click save.

Gradescope submission video here: https://www.youtube.com/watch?v=KMPoby5g_nE
PDF submission guide here (ignore the parts about scanning): https://gradescope-static-assets.s3.amazonaws.com/help/submitting_hw_guide.pdf

Setup

Back on your lab computer, navigate to your home directory from the icon on the Desktop, and create a ***working directory*** for the lab called lab01 (you may want to put this in an EAPS577 folder). Download all of the files for the lab from this folder into your working directory: https://www.dropbox.com/sh/xsogyz0oylstjmb/AABu-R9Ogth5pvRPRvYcYhtga?dl=0

I recommend unzipping the file in your downloads folder and then copying to your working directory, much faster.

There are two key documents for this lab - this document (lab01-instructions.pdf) gives instructions for the lab, but make sure to also fill out the lab report (lab01-report.doc). You should open and edit this file, and write your answers in all the indicated areas.

Anytime you are working in IDL, you should start a journal file, as discussed in the tutorial. These files should be saved in your working directory, and are useful in case we have questions about whether you can demonstrate that you generated the images below yourself. If you can't get this working, let us know.

Intro: Pixels and Colors

Your display screen consists of about a million little areas called **pixels**. Each pixel can show a different color/intensity combination. Everything on your screen - text, pictures, or whatever - is displayed by filling all the pixels with the appropriate color/intensity. Your screen dimensions are large (my 13" MacbookPro is 2560x1600), so each pixel is very small.

All colors seen by the human eye can be produced by a mixture of intensities of only three colors: red, green, and blue (RGB). These colors are similar to the wavelengths detected by the cones at the rear of our eyes. Most displays in use today allow 256 intensities of each color. This gives rise to 256^3 RGB combinations and thus that many unique colors - a big number (often referred to "true color" or "millions of colors"). However, some screens can't display this full range of colors. Instead, they can display only 256 different combinations. This may seem small, but for most purposes it's exactly what you need. For example, in a standard TIFF or JPEG greyscale image there can be 256 different intensity levels, ranging from black, through the grey scales, to the brightest white.

Key Concepts in this Lab:

- Stretching images to show variability
- True color vs. False color images – how to make them and why each is useful
- Creating and debugging a simple IDL function

1. Stretching Images

Make sure that you are working inside of your new working directory by using the "current directory" field in IDLDE.

The first image that we'll work with this week is an image (w20140910t183501774id30f17) taken in 2014 by the OSIRIS Wide Angle Camera onboard the Rosetta spacecraft. The image shows the comet 67P-Churyumov-Gerasimenko at about 27 km away during the global mapping phase portion of the mission. The **cg_img.sav** file contains the data saved as an idl variable, and

it should now inside of your working directory for this lab.

This file was created using the **save** command in IDL (discussed in the IDL tutorial last week), so type the following **restore** command to retrieve the variables:

```
IDL> restore,'cg_img.sav'
```

The image is represented as a two-dimensional array. What are the array dimensions? To find out, type:

```
IDL> help
```

To find out the range of values in the image, use the **min** and **max** commands to return the smallest and largest elements of the array, respectively:

```
IDL> print, min(cg_img), max(cg_img)
```

What is the range of the data values in the image? The image has been calibrated, so the units are in radiance (W/m^2/steradian/nm). We'll talk about what this means later in the term. (Question #1a in the Lab Report)

The most common program that I use to look at images in IDL is called ATV2, which should be in your working directory. Let's try to display the data using ATV2 (you also have ATV which was throwing some bugs later on in the lab, but feel free to try both!):

```
IDL> atv2, cg_img
```

The "min" and "max" fields in the atv window show the current stretch applied to the image. *"Stretching" an image is choosing which data values should be displayed as bright (white) and dark (black).* Because a typical greyscale image can contain up to 256 colors, this means choosing which data values are displayed as 255 (white) and 0 (black). ATV has chosen some limits automatically, as shown, which in this case correspond to the min and max values.

From the ATV menu bar, change the scaling (stretch) type to "Log", and save the image in ATV: *File -> WriteTIFF*. Save it as *cgimg_log.tif* in your lab01 directory and paste or insert it into your lab report. (Question #1b in the Lab Report)

This type of stretch is non-linear – changes in brightness in the image are modified from what we would see in real life. You can see what an asinh function looks like by plotting it like this, where "alog10" is IDL for log base 10:

```
IDL> window,0, xs=600, ys=400
IDL> plot,alog10(findgen(256)), xtit='Bytes', ytit='Relative
Displayed Brightness'
```

Here, the x-axis corresponds to the actual data values in the image, and the y-axis corresponds to the value that they're displayed at (the stretch). In which part of the x-axis does the curve change most rapidly? Based on your answer, does this stretch emphasize variability in the dark or bright parts of the image? (Question #1c-d in the Lab Report)

For scientific interpretation, a linear stretch is usually a little more intuitive. To change to a linear stretch in ATV2, select "linear" under the "scaling" menu. Now the change in brightness

values between different regions of the image is true to the observation. Change the min and max values until you can see the most detail in the image with this stretch.

Save your favorite linear stretch of the image in ATV2: *File -> WriteTIFF*. Save it as *cgimg_linear.tif* in your lab01 directory and paste or insert it into your lab report. (Question #1e in the Lab Report)

For this observation, which image do you prefer (linear vs. log), and why? Does one allow you to see features that the other does not? (Question #1f in the Lab Report)

2. Making a true color image

The data we are going to use in the second part of this lab is a Hubble Space Telescope image of Mars taken on May 13, 2001 at three different wavelengths: 410 nm (blue), 502 nm (green) and 673 nm (red). The **mars_colorimg.sav** file contains the data in three different IDL variables (*redimg*, *grnimg* and *bluimg*), and it is now in your lab01 directory. Restore the data:

```
IDL> restore,'mars_colorimg.sav'
```

To find out the range of values in each image, use the **min** and **max** commands to return the smallest and largest elements of the array, respectively:

```
IDL> print, min(redimg), max(redimg)
IDL> print, min(grnimg), max(grnimg)
IDL> print, min(bluimg), max(bluimg)
```

What is the range of the data values in the *redimg*, *grnimg* and *bluimg*? These images have been calibrated to units of albedo, or percent reflectance of sunlight at that wavelength. Which one has the highest value, and what is the max value? If this was a similar image of Earth, which image would have the highest value and why? (Question #2a-b in the Lab Report)

Take a look at the red image in atv, and try both a linear and an asinh stretch.

```
IDL> atv2, redimg
```

To make a color composite, we need to place these images into red, green, and blue channels, but if we want to display the correct color composite of these three images, we need to correctly scale the images. To do this, we will use the **bytscl** command, which transforms the images from numerical albedo into 0-255 scaled images. The units are now in "bytes", which are the simplest record that you can store in computer memory, and are integers that can range from 0-255. Try this out using **atv2**:

```
IDL> atv2, bytscl(redimg)
```

Now the image is stretched between 0 and 255, and only has integer values. You can use bytscl to specify the min and max of the stretch as well.

```
IDL> atv2, bytscl(redimg, min=0., max=max(redimg))
```

The **bytscl** function scales all values of the array into the range 0 to 255 where 0 corresponds to the value set with the **min** keyword and 255 corresponds to the value set with the **max** keyword. If the **max** keyword is not provided, then the maximum value of the array is set to 255,

and if the **min** keyword is not provided, then the minimum value of the array is set to 0. So the second command above is equivalent to the first.

To create a "true" color image, we need to scale all of the arrays to the same limits and use a linear scale. You should know what the maximum value of all of the arrays is from Question #2a, so now let's set a variable equal to that number:

```
IDL> max_value = (your max value here)
```

Now create scaled versions of all of the images using the same min/max values:

```
IDL> mr = bytscl(redimg, min=0., max=max_value)
IDL> mg = bytscl(grnimg, min=0., max=max_value)
IDL> mb = bytscl(bluimg, min=0., max=max_value)
IDL> print, max(mr), max(mg), max(mb)
```

From the output of the last command, you see that the arrays span different ranges, as they should. To display the composite, we'll use the **tv** command, which is the simplest way to look at images in IDL. But first we need to make a window to output the image to. The image is 460 x 460, so let's make the window that size as well:

```
IDL> window, 0, xsize=460, ysize=460, title='HST Mars 2001/05/13'
```

The 0 creates the new window number 0, which appears in the upper right corner of the screen (you could also use /free instead of 0, which creates a new window number and leaves any previous graphics window(s) intact). The size of the window is determined by setting the **xsize** and **ysize** keywords to the specific width and height you want in pixels. Finally, the title keyword lets you personalize the window's label.

Now output the images to the display using the **tv** command. We can use the **channel** keyword to manipulate the **tv** command to show red (channel=1), green (channel=2) or blue (channel=3). Look at each channel individually first, using the erase command to clear the display between images:

```
IDL> tv, mr, channel=1
IDL> erase
IDL> tv, mg, channel=2
IDL> erase
IDL> tv, mb, channel=3
IDL> erase
```

Now combine all three, either by running all three tv commands above in sequence without the erase command, or just by using this one command with the true keyword set:

```
IDL> tv, [[[mr]], [[mg]], [[mb]]], true=3
```

Figure 1 shows what each image should look like.

Before moving further in this tutorial, let's save the three byte image arrays:

```
IDL> save, mr, mg, mb, filename='mars_byteimages.sav'
```

Check that this *mars_byteimages.sav* file is created in your working directory. If you want to reopen this data later, use:

```
IDL> restore,'mars_byteimages.sav'
```

Output your image as a TIFF using the **tiff_write** procedure.

```
IDL> tiff_write, 'hst_mars_truecolor.tif', red=mr, green=mg, blue=mb,
planarconfig=2
```

Check that this *mars_truecolor.tif* file is created in your working directory, and paste or insert it into your lab report. Question #2c in the Lab Report)

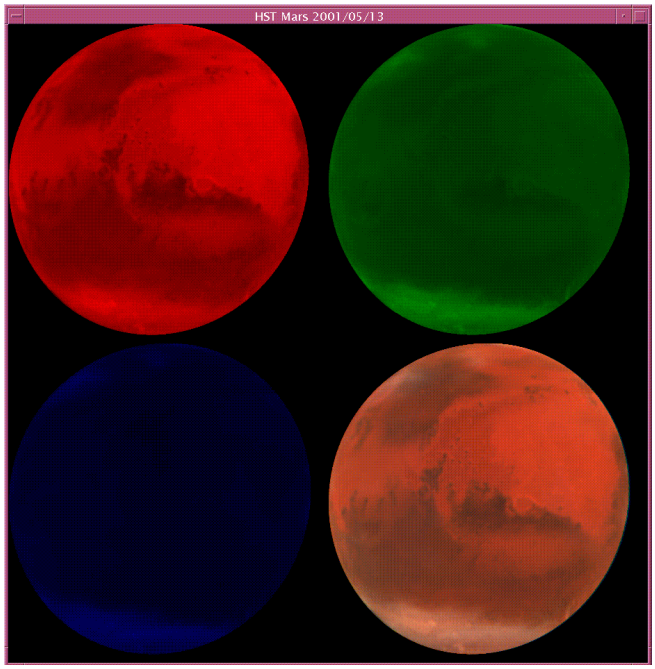

Figure 1 *– HST image of Mars, RGB channels and approximate true color.*

3. Making a false color image

As we talked about in class, most images we see of solar system bodies aren't true color, because true color doesn't always emphasize variability in an aesthetically pleasing (or scientifically useful) way. For this part of the lab, I want you to create a false color image in the same way that you created a true color image above, but ***aim to emphasize the ice and clouds in the image instead***. Think about what you have to change to accomplish this. What stretch limits did you use to create your false color image showing clouds and ice on Mars? (Question #3a)

When you have your false color image, output it as a TIFF called '*hst_mars_falsecolor.tif*', as above, and paste it into your lab report. (Question #3b)

4. Create a program to write out true color images

Now let's automate this process within an IDL program. ***We'll discuss IDL programming basics in class, and see the slides on Brightspace for key info to help you complete this section.***

The purpose of the program will be to take any 3-band color image and save it out as a tif as you did at the command line above. In IDLDE, create a new .pro file in your lab01 directory called **`write_true_color.pro`**. I've also provided a template that you can use for the program if you like, **`write_true_color_template.pro`**, in the shared folder. Make sure to rename this file and add your name/date to the comments at the top. Note that this code is written as a program instead of as a function – why is this appropriate? (Question #4a)

The first line of the file should start with the following line to define the program and the names of the input variables. In this case, the input variables will be the RGB portions of the image you want to write out and the name of the image file:

```
PRO write_true_color, red, grn, blu, filename
```

In the program, you don't have to refer to the specific image (e.g., mr), you can just refer to the generic variable (e.g., red). The last line in the file should be an END statement:

```
END
```

In between these lines you should create and write out the true color image. Please make sure to comment as needed. The template includes some basic comments but you should add your own wherever you need to – these notes are both for future you and anyone else who runs the code.

You should be able to copy and paste the command line code you wrote to output an RGB true color tiff of the Hubble image, changing the variable names. If you want to get some feedback from your program as it runs, you can add another line after this that prints to the screen:

```
print,'Created RGB image: ' + filename
```

Compile your code and then test it using the HST data above. If your image outputs as a weird colorful rainbow of pixels, you forgot to bytscl it first. If IDL hits a bug, it will stop and tell you where something is wrong. Fix the error, compile, and try again. When you have working code, please copy and paste it into your lab report (Question #4b), then paste the resulting true color HST image (suggested name: **`hst_mars_true_color_auto.tif`**) into your report as well (Question #4c).

5. Create a program to write out false color images

Now we'll repeat the process to create code to write a false color image. "False" in this case will be defined by you. What type of stretch will you use to create your false color image, noting that we'll use this same program for images of a variety of targets (not just Hubble Mars), and why? (Question #5a) Some options to consider:

- Linear stretch: stretch each channel from its min to max
- Linear 2%: stretch each channel from 2% to 98% of the max value (cut off the tails)
- Square root: Enhance variability at the high end by taking the square root of the data before scaling
- Logarithmic stretch: Enhance variability at the low end by taking the square root of the data before scaling

(Extra: if you want to be fancy you could code multiple options and then select via an input flag, look up how to use the"keyword_set" function along with if/then statements to do this)

Make a copy of your program and rename it **`write_false_color.pro`**. Modify the code to produce a false color image to your specifications outlined above (suggested name: **`hst_mars_false_color_auto.tif`**). When you have working code, please copy and paste it into your lab report. (Question #5b), then paste the resulting false color HST image into your report as well (Question #5c). Are you happy with your stretch? What does it do well vs. poorly? How does it compare to the custom stretches above? (Question #5d).

Congratulations, you can program in IDL!

Parts of this were written by Min Y. H. Hubbard and Jim Bell; later added to, updated, and modified by Briony Horgan.