Chapter 14

Numpy arrays and image manipulation

For this week's assignment you will do some simple image manipulations and plots of two-dimensional arrays. The goals for this week are to:

- create and resize Numpy arrays,
- introspect size and data types,
- use indexing and slicing for arrays,
- use `imshow` to show two-dimensional arrays,
- do some simple array manipulations.

Be sure to stick to the PEP 8 style rules, and do not forget to properly label any plots you create.

14.1 Creating a simple one-dimensional plot

- For this part of the assignment create a Python script called `planck.py`.
- Make a plot of the $B_\lambda(\lambda, T)$ Planck curve [1]
- Use `numpy.linspace` to create an array of wavelength values.
- Implement Planck equation in terms of `numpy` math functions that operate on arrays.
- Add the plot to your report.

14.2 Image manipulation

14.2.1 Loading the image

- Download the image provided on the course website, you will use this image throughout the assignment. Create a new Python script and call it `imagemanipulations.py`, in that script create a function texttttranspose_image where you add this section's code (to test call that function). This function takes one argument, the name of the file that should be transposed.
- As a first step you are going to load the image and turn it into a Numpy array. There are several functions you could use to perform this step, but we will use `scipy.ndimage.imread` . Look up the documentation for this function and load the image into an array.
- For your report: what are the dimensions of this image and what are the data types in the array?

[1] See for instance `https://en.wikipedia.org/wiki/Planck's_law`.

- Use `pyplot.imshow` to show the image using Python. Transpose the X and Y axes of the image using the `numpy.swapaxes` function. Include the transposed image in your report.

14.2.2 Making a color separation

Images as displayed on your screen can be separated in a red, a green, and a blue channel.

- Start by creating function `color_separation` for this section of the assignment. This function should take one argument, the filename of the image.
- Load the image into an array.
- Create a grid of four sub plots using the `pyplot.subplots` function. In the top left plot put the original image, use the other sub plots to show the individual color channels.
- The image consists of a red, a green and a blue color channel. Use array indexing / slicing to create arrays for the individual color channels. Add the individual channels to the other three sub plots with appropriate color maps (look up the `cmap` argument for `pyplot.imshow`).
- Remove the pixel coordinates from the axes as they are not relevant.
- Add the plot to your report.

14.2.3 Creating a gray scale image

You will perform that separation and, later, create a gray scale image using the following equation[2]

$$Y = 0.299R + 0.587G + 0.114B,$$

where R is the red channel value, G the green channel value, and B the blue channel value.

- Create a function `grayscale` that takes one argument (the filename of the image to make a grayscale version of).
- Load the image into an array.
- Gray scale images have the same value for each of the red, green and blue channels. You can create a single array for a gray scale image and use an appropriate color map to show it as a gray scale image. The other way is to create identical red, green, and blue channels and use `pyplot.imshow` to show that three channel image.
- Add the grayscale image to your report.

14.2.4 Detecting edges using a Sobel filter

In this section you will create a Sobel filter to detect edges in the image. Note you are not allowed to use `scipy.ndimage.filters.sobel` for this part of the assignment. Read the Wikipedia page on Sobel filters[3]. *This part the exercise is a bit harder than the earlier parts.*

- Create a new function `sobel_filter` that take a single argument (the filename of the file to operate on).
- Create a two-by-two grid of subplots (using `pyplot.sublots`).
- You can use array manipulations combined with the `scipy.signal.convolve` to implement the Sobel filter.
- Show the original image at the top left. The x-gradient on the top right, the y-gradient on the bottom left and the result of the complete Sobel filter at the bottom right.
- Make sure to properly label these plots.
- Add the resulting image to your report.

[2] With Y the luminance see `https://en.wikipedia.org/wiki/Grayscale`

[3] See `https://en.wikipedia.org/wiki/Sobel_operator`.

14.3 Hints

- If you are having problems with plots getting drawn on top of each other use the `pyplot.clf` function to clear the older plots.