Python Second Exercise

Gines Gonzalez Guirado

October 2023

1 Explanation of the code.

First of all, I have imported the libraries that I am going to use. Then, I set the functions that I have made for this task in order of use and with descriptive names.

1. The function `open_and_check_files_existence(v_file: str, i_file: str)` is used to open and check the existence and the path .fits files. If the path of any or both files is wrong or they do not exist, it raises an error. If everything is correct, it opens the files and checks that the dimensions of the images are the same as each other and with those that appear in each header in the entry "NAXIS1" (size on the x axis) and "NAXIS2" (size on the y axis). Finally, it returns the images and the header of each image.

2. The second function `sum_images(image_1,image_2)` is for the second section and it is to add the images V and I. It sums the elements of the arrays that are in the same position. It gives an error if the dimension of the final image do not match with the dimensions of the images that are added, this indicates that the sum has been done bad.

3. The next function, `most_brighter_pixel(img)`, is also for the second section and it is to obtain the brighter pixel and its position. It uses `np.nanargmax` to find the index of the brightest pixel, but `np.nanargmax` gives the index of the maximum value of an array as whatever its position in an 1D array. Then, `np.unravel_index`

converts, by introducing the shape of the matrix with `.shape`, the index of this pixel into row and column indices to determine the position of the brightest pixel in the matrix. Finally, it obtains the value of the brightest pixel, because we now know its position in the image. It returns the value of the brightest pixel and its position.

4. The function `open_mask_and_multiply_by_image(mask: str, image)` is a function that opens and check the existence and the path of the mask. Then, it obtains the background (noise) of the image by multiplying the image by the mask. It returns the background of the image `(noise_image)` and the mask `(mask_image)`.

5. The function `calculate_mask_coverage_with_nan(mask)` is a function to show the total percentage of the image that is cover by the mask. It is not compulsory for the task. Firstly, it calculates the total number of NaN pixels that are contained in the mask. Then, it obtains the total number of pixels of the mask. Finally, it calculates the percentage of NaN pixels that cover the mask and returns it.

6. The function `measure_noise_with_random_apertures(noise_image, num_apertures, aperture_shape)` is a function to measure the noise of the astronomical image, using 10 apertures of $10x10$ pixels randomly distributed in the image after applying the mask. I have set the condition that the NaN pixels must be less than the 40% of the total aperture. This is to obtain a better value of the real noise of the astronomical image.

   First of all, it creates an array to store the noise measurements for each aperture. Then, it inicializes all values as NaN. Next, it stores the number of rows (height) and the number of columns (width). Now, I have created a while loop to randomly make apertures and save their standard deviation in `noise_measurements`, when the NaN pixels in each aperture are less than 40% of it.

   So, in the while loop the following happens: Firstly, it generates random coordinates for the top-left corner of the aperture. Then, the aperture is defined as the matrix with the elements from `top_left_y` to the `top_left_y` column plus the number of columns of the matrix and it is done in an analogous way for the x axis. Now, it calculates the total number of NaN pixels that are contained in the aperture. It

also calculates the total pixels of the aperture. Then, it checks if the percentage of NaN values exceeds 40%. If the NaN pixels in each aperture are less than the 40%, it calculates the standard deviation and stores it in `noise_measurements`, and it moves to the next aperture. If the NaN pixels in each aperture are not less than 40%, the while loop is repeated without storing any value in `noise_measurements`. Finally, it calculates the average noise value and returns it.

7. Now I have defined the files that I want to open with the functions that I have done. I have not entered the path because the files are in the same folder as the program and, in this case, the path does not have to be specified. But for it to work if they are in separate folders you have to set the path correctly.

8. Now, I call the function `open_and_check_files_existence(v_file: str, i_file: str)` to open and check the files existence and storing the return values in variables. Then, I represent both images with *matplotlib* and the code provided by Fernando Buitrago in class.

9. Next, I call the function `sum_images(image_1,image_2)` to add the images and storing the return value in variable (image). Then, I represent the image in the same way.

10. Now, I call the function `most_brighter_pixel(img)` to find the brighter pixel, assuming the image is an array and storing the return values in variables and printing them on the screen.

11. Afterwards, I call the function `open_mask_and_multiply_by_image(mask: str, image)` to multiply the image by the mask and storing the return values in variables. Then, I call the function `calculate_mask_coverage_with_nan(mask)` to calculate the image percentage covered by the mask and it is printed on screen. I also represent the noise image in the same way as the other graphical representations.

12. Finally, I define the number of aperture, 10, the shape of the matrix that represents the aperture, (10,10), and I call the function `measure_noise_with_random_apertures(noise_imag num_apertures, aperture_shape)` to create the apertures and calculate the average noise. The result is printed on screen.

2 Problems when doing the code and decisions.

The first problem was when I wanted to check the existence of the files and I found *os.path.isfile* on internet, though I think that we have seen it in class.

The next problem was to find the most brighter pixel. Firstly, I thought about doing it with a double loop going through all the pixels of the image, after having found the maximum value with *np.nanmax*. But, I searched the internet for another more efficient way to do it and found *np.unravel*, which does the opposite of *np.ravel* that we had seen in class.

The most difficult part was to do the apertures, with less than the 40% of NaN values in each aperture, to find the noise. In order for all apertures to contribute by averaging their standard deviations, I had to make the while loop so that their standard deviation values were only stored in `noise_measurements` when that condition was achieved.

I have separated things that could go on one line into two lines with backslash(\) so that the 80 characters on one line would not be exceeded.