Universidade Federal do Rio de Janeiro
Centro de Ciências da Matemática e da Natureza
Observatório do Valongo

Tratamento de Dados - OVL474

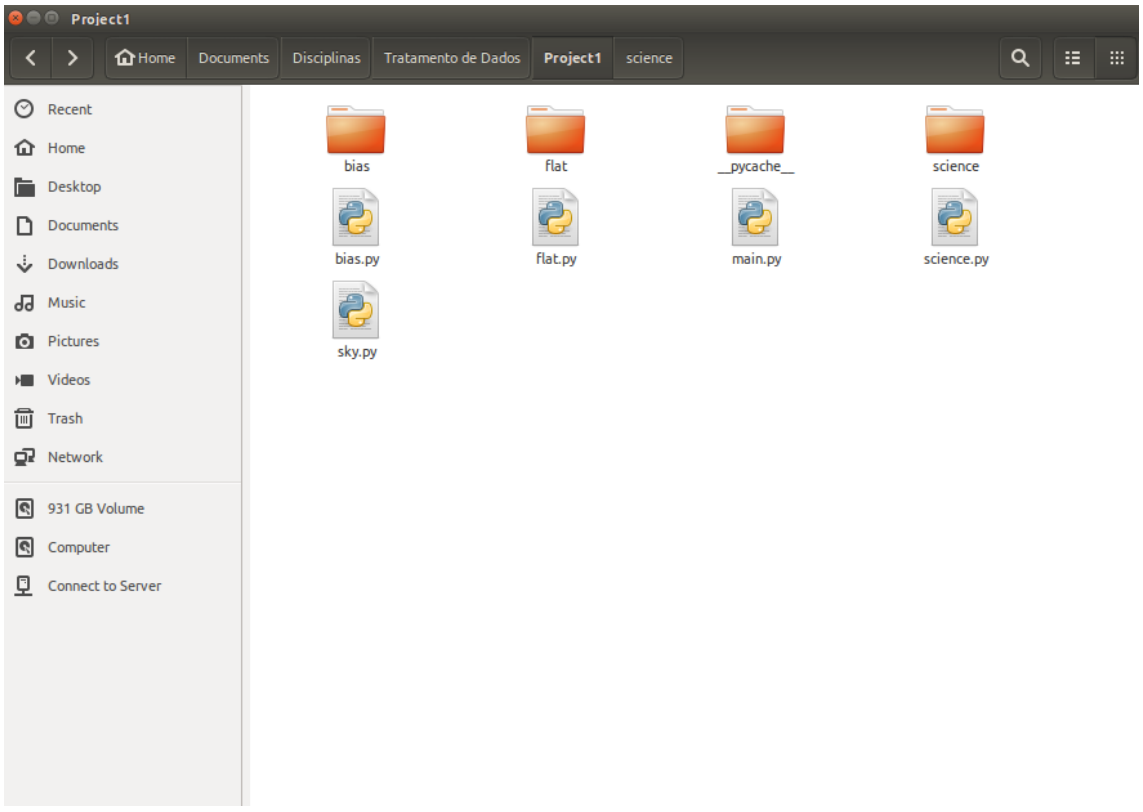
Pipeline para Redução de Imagens

Por: Natália Nogueira Maia

15 de Setembro de 2017

Introdução

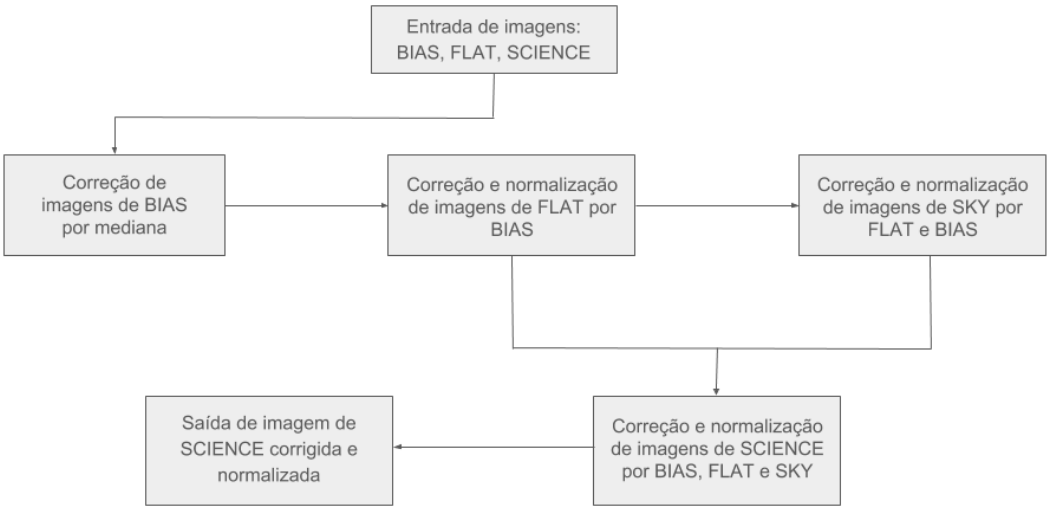
Este programa é usado para redução de dados em imagens observacionais, levando em consideração os erros de BIAS, FLAT e SKY. Note que para o funcionamento perfeito do mesmo, é necessário que cada uma das imagens esteja em suas devidas pastas dentro do diretório em que todos os códigos-fonte *bias.py, flat.py, science.py, sky.py, main.py* se encontram, como mostrado na imagem a seguir:



Neste programa, estamos supondo que o usuário está reduzindo os dados de apenas uma noite de observações, e que esta noite se trata de uma noite fotométrica, sem grandes diferenças no céu. Por isso, as coordenadas pedidas para a correção de SKY se referem somente à primeira imagem. A entrada deve ser feita somente com números inteiros, de outra forma o programa não funcionará.

Funcionamento do Código

O código funciona seguindo a lógica dada no fluxograma a seguir:




```
10  def main():
11
12      # Getting all image names:
13
14      bias_tables = glob.glob("bias/*.fits")
15      flat_tables = glob.glob("flat/*.fits")
16      science_tables = glob.glob("science/*.fits")
17
18      # Getting limits for SKY correction in first SCIENCE image:
19
20      sky_inferior_abscissa_limit = input(
21          "Please input inferior abscissa limit for sky correction based on first SCIENCE image: \n")
22      sky_superior_abscissa_limit = input(
23          "Please input superior abscissa limit for sky correction based on first SCIENCE image: \n")
24      sky_inferior_ordinate_limit = input(
25          "Please input inferior ordinate limit for sky correction based on first SCIENCE image: \n")
26      sky_superior_ordinate_limit = input(
27          "Please input superior ordinate limit for sky correction based on first SCIENCE image: \n")
28
29      first_science_image = science_tables[0]
30
31      # Calling each function to correct SCIENCE images for BIAS, FLAT and SKY:
32
33      bias_correction = bias(bias_tables)
34      bias_flat_correction = flat(flat_tables, bias_correction)
35      sky_correction = sky(first_science_image, bias_correction, bias_flat_correction, sky_inferior_abscissa_limit,
36                           sky_superior_abscissa_limit, sky_inferior_ordinate_limit, sky_superior_ordinate_limit)
37
38      bias_flat_science_images = science(science_tables, bias_correction, bias_flat_correction, sky_correction)
```

Para correção de BIAS, simplesmente tiramos a mediana de todas as imagens no diretório **bias**. Isso nos dá uma matriz, *bias_correction*, que utilizaremos para corrigir as imagens subsequentes.

```
5   def bias(bias_tables):
6       bias_images = bias_tables
7
8       # Opening data for each BIAS image:
9
10      images = []
11      for value in bias_images:
12          image, header = fits.getdata(value, header=True)
13          image = np.array(image, dtype=np.float)
14          images.append(image)
15
16      # Combination of BIAS images:
17
18      bias_correction = np.median(images, axis=0)
19      print(bias_correction)
20
21      # Freeing system memory:
22
23      del bias_images
24      del images
25
26      return bias_correction
27
```

Para correção de FLAT, primeiro subtraímos *bias_correction* de todas as imagens no diretório **flat**. Chamando as matrizes resultantes de *bias_flat_images*, tiramos sua média e operamos

$$bias\_flat\_images = \frac{bias\_flat\_images}{media(bias\_flat\_images)} \tag{1}$$

de forma a normalizar todas as nossas imagens de FLAT corrigidas por BIAS. Em seguida, tiramos a mediana dessas imagens, resultando na matriz *bias_flat_correction*.

```
def flat(flat_tables, bias_correction):
    flat_images = flat_tables

    # Opening data for each FLAT image:

    bias_flat_images = []
    for value in flat_images:
        image, header = fits.getdata(value, header=True)
        image = np.array(image, dtype=np.float)

        # BIAS correction applied to FLAT images:

        bias_flat_images.append(image - bias_correction)

    # Normalization of BIAS corrected FLAT images:

    bias_flat_images = bias_flat_images/np.mean(bias_flat_images, axis=0)

    # Combination of BIAS corrected and normalized FLAT images:

    bias_flat_correction = np.median(bias_flat_images, axis=0)
    print(bias_flat_correction)

    # Freeing system memory:

    del flat_images
    del bias_flat_images

    return bias_flat_correction

```

A partir daí, partimos para a correção de SKY. Para tal, retiramos uma porção da primeira imagem SCIENCE para esta correção. Devemos também redimensionar nossas imagens de correção de BIAS e FLAT para os pontos que escolhemos para SKY. Para tal, criamos duas novas matrizes, *sky_bias_correction* sendo a matriz de BIAS no local definido para SKY, e *sky_flat_correction* sendo a matriz FLAT para o mesmo local. Em seguida, aplicamos

$$sky\_correction = \frac{sky\_image - sky\_bias\_correction}{sky\_bias\_flat\_correction} \tag{2}$$

tendo, assim, a matriz *sky_correction* com os valores de SKY corrigidos por BIAS e FLAT. Desta matriz, tiramos sua mediana para aplicá-la às imagens SCIENCE.

```
5   def sky(first_science_image, bias_correction, bias_flat_correction, sky_inferior_abscissa_limit,
6           sky_superior_abscissa_limit, sky_inferior_ordinate_limit, sky_superior_ordinate_limit):
7
8       # Opening first SCIENCE image to measure the SKY:
9
10      image = fits.open(first_science_image)
11      sky_data = image[0].data
12
13      # Delimiting portion of first SCIENCE image that gives us the SKY:
14
15      inferior_ordinate_limit = int(sky_inferior_ordinate_limit) - 1
16      inferior_abscissa_limit = int(sky_inferior_abscissa_limit) - 1
17      superior_ordinate_limit = int(sky_superior_ordinate_limit)
18      superior_abscissa_limit = int(sky_superior_abscissa_limit)
19
20      sky_image = sky_data[inferior_ordinate_limit:superior_ordinate_limit,
21                           inferior_abscissa_limit:superior_abscissa_limit]
22
23      # Sampling the same portion of BIAS correction and FLAT correction to apply to SKY image:
24
25      sky_bias_correction = bias_correction[inferior_ordinate_limit:superior_ordinate_limit,
26                                            inferior_abscissa_limit:superior_abscissa_limit]
27      sky_bias_flat_correction = bias_flat_correction[inferior_ordinate_limit:superior_ordinate_limit,
28                                                      inferior_abscissa_limit:superior_abscissa_limit]
29
30      # BIAS and FLAT correction applied to SKY images:
31
32      sky_correction = (sky_image - sky_bias_correction)/sky_bias_flat_correction
33
34      # Combining values of SKY correction:
35
36      sky_correction = np.median(sky_correction)
37
38      return sky_correction
39
```

Finalmente partimos para as imagens SCIENCE. Lembrando que todas as imagens devem estar no diretório **science**, fazemos a seguinte operação

$$bias\_flat\_science\_image = \frac{science\_images - bias\_correction}{bias\_flat\_correction} - sky\_correction \tag{3}$$

para corrigir por BIAS, FLAT e SKY.

```
def science(science_tables, bias_correction, bias_flat_correction, sky_correction):
    science_images = science_tables

    # Opening data for each SCIENCE image:

    bias_flat_science_images = []
    for value in science_images:
        image, header = fits.getdata(value, header=True)
        image = np.array(image, dtype=np.float)

        # BIAS, FLAT and SKY correction applied to SCIENCE images:

        bias_flat_science_images.append(((image - bias_correction) / bias_flat_correction) - sky_correction)

    # Freeing system memory:

    del science_images

    return bias_flat_science_images

```

Após é feita uma pergunta ao usuário se gostaria de combinar todas as imagens de ciência disponíveis ou acessá-las uma a uma. No caso de combinação, usamos da mediana de todas as imagens para combiná-las. As imagens resultantes são frutos deste processo aliados à estatísticas para uma melhor qualidade das mesmas.

```
    # Ask the user if he wants to combine all SCIENCE images:

    ask = input('Want to combine all SCIENCE images [yes/no]: ')

    if ask == 'yes':
        bias_flat_science_combination = np.median(bias_flat_science_images, axis=0)

        # Plot of single SCIENCE image:

        image = bias_flat_science_combination
        image.flatten()
        np.median(image)
        np.std(image)
        np.mean(image)

        plt.figure()
        plt.imshow(image, vmin=np.mean(image) - 2.5 * np.std(image), vmax=np.mean(image) + 2.5 * np.std(image),
                   cmap=plt.get_cmap('Greys'))
        plt.colorbar()
        plt.show()

    else:

        # Plot each SCIENCE image:

        for image in bias_flat_science_images:
            image.flatten()
            np.median(image)
            np.std(image)
            np.mean(image)

            plt.figure()
            plt.imshow(image, vmin=np.mean(image)-2.5*np.std(image), vmax=np.mean(image)+2.5*np.std(image),
                       cmap=plt.get_cmap('Greys'))
            plt.colorbar()
            plt.show()

main()

```

Observações

O código não se encontra da forma mais otimizada, pois não sei como fazê-lo. Ao reduzir muitas imagens SCIENCE ao mesmo tempo, temos problema de memória RAM do computador, nos impossibilitando de seguir adiante com o trabalho, então é recomendável que se faça essa redução em partes.