From 3D cubes to 2D images

Edgar Ortiz
edgar.ortiz@ua.cl

Abstract

In this work, I present a python script (attached) developed to compute photometric data from spectral data cubes (images where each pixel is a spectrum). The photometric data is computed using either a given filter as those used in telescopes or a perfect rectangular filter in a given wavelength range. To accomplish that, the fluxes for each pixel are integrated over the wavelength range of the filter provided, weighted by the filter itself (1). The arguments for the script are:

1. The path of the data cube file and the path of the filter file, or
2. The path of the data cube file and the two wavelengths, lambda1 and lambda2 such lambda1 < lambda2. These last two arguments are the wavelength range needed in order to create a perfect rectangular filter.

The output of the script corresponds to a fits file with the photometric data for one-ninth of the data in the data cube. This parameter must be modified directly on the script. This was done because of the memory limitation in the computing power I had access to.

Introduction

With the current technology in astronomy, now it is possible to produce more than just photometric data. Now it is possible thanks to IFUs (Integral Field Units) and radio interferometers to produce spectral cubes, that is, an image where each pixel is a spectrum. By integrating the spectroscopy data, we can always recover photometric data, since photometric data can be regarded as low-resolution spectroscopy. To do that, we integrate the fluxes of the spectrum in a given wavelength range. If we want to compare that computed photometric data with the one obtained in a telescope using a given filter, we have to weight the fluxes by the filter, according to the next equation:

$$f_\lambda(T) = \frac{\int T(\lambda) \times f_\lambda(\lambda) d\lambda}{\int T(\lambda) d\lambda} \tag{1}$$

Here *T* refers to the filter.

Data

In this work, we compute the photometric data of our spectral cube in two filters (attached) and in two wavelengths ranges:

- i_prime.dat → 660 nm to 880 nm
- r_prime.dat → 530 nm to 730 nm
- 674 nm to 681 nm
- 683 nm to 689 nm

For all the four cases above, we normalize the integral of the filter to one such that when we are computing (1), we compute only the numerator. In the last two cases, the filter is constant over the wavelength range provided and zero outside of it, in this case, the normalization is such that the constant value of the filter is one divided by the length of the wavelength range.

Algorithms

thD2twD.py: is a python script that computes the photometric data for our data cube. There are two possibilities for the arguments of the script:

1. python **thD2twD.py** path_to_the_data_cube path_to_the_filter
2. python **thD2twD.py** path_to_the_data_cube lambda1 lambda2

The arguments lambda1 and lambda2 are the wavelength range over which we want to compute the flux and the following condition must hold: lambda1 < lambda2.

The script computes the photometric data and saves it as a fits file. The script uses the following classes and functions to accomplish that task:

1. **Cube_handler()**: this class manages the data cube. It is initialized with the first argument of the script: **cube**, a variable named **test**, set by default to True, which indicates that the photometric data will be computed only on one-ninth of the slices in the data cube. There is also a variable **n**, set by default to 1 that indicates on which of the slices' subsets the photometric data will be computed. This class also contains the next methods:
    a. **u_convert()**: returns the conversion factor in order to convert the units of the flux to 'J/(nm m2 s)'
    b. **cube()**: returns a 3D array with the fluxes converted to 'J/(nm m2 s)'
    c. **lamb_s()**: returns a 1D array with the wavelengths of each slice in the data cube.
    d. **interpolate()**: it receives an interval over which the interpolation of the fluxes in the data cube is evaluated. This evaluation is returned by this method.
    e. **close()**: this method closes the fits file that was opened using astropy.io.
2. **Filter_handler():** this class manages the data in the filter and it is initialized with either the second argument of the script or the second and the third, depending on how the script is called (see the beginning of this section). It contains the next methods:
    a. **lamb_f():** returns a 1D array of wavelengths in nm

    b. **energy():** it returns T (1) in units of energy, such that the integral of T is normalized to one.
    c. **interpolate():** it receives an interval over which the interpolation of the filter data is evaluated. This evaluation is returned by this method.

3. **lamb_inter(arr_1,arr_2)**: returns the sorted union of the 1D arrays it gets as input. Additionally, the elements that are repeated, are striped, letting only one.
4. **image(**lambdas,filter_energy,cube_flux,n=1,filter_name=None,lambda1=None, lambda2=None**):** writes to the hard disk a fits file containing the image generated using (1).
    a. **lambdas:** is the union of the wavelength interval of the filter and the data cube.
    b. **filter_energy:** is the energy of the filter computed on the **lambdas** interval.
    c. **cube_flux:** is the flux of the data cube computed on the **lambdas** interval.
    d. **n:** is the set of slices upon which the photometric data is computed.
    e. The other ones speak by themselves

The implementation of the algorithm is:

```
# Loading cube and filter
i = 8 # set of slices to compute the photometric data
cube   = Cube_handler(cube_name,test=True, n = i+1)
if n_arguments == 3: # testing if the script arguments contains either the
filter or the wavelength interval
   filter = Filter_handler(filter = filter_name)
else:
   filter = Filter_handler(lambda1 = lambda1,lambda2=lambda2)
# lambdas: 1st argument for image()
filter_lambdas   = filter.lamb_f()
```

```
cube_lambdas   = cube.lamb_s()
lambdas           = lamb_inter(filter_lambdas,cube_lambdas)
# Filter energies and cube fluxes
filter_energy = filter.interpolate(lambdas)
cube_flux     = cube.interpolate(lambdas)
# generating the image
if lambda1: # testing if the script arguments contains either the filter or the wavelength interval
        image(lambdas,filter_energy,cube_flux,n=i+1,  lambda1  =  lambda1, lambda2 =lambda2)
else:
    image(lambdas,filter_energy,cube_flux,n=i+1,filter_name=filter_name)
cube.close()
```

Discussion

The main difficulty for the creation of this script was not being familiar with astropy (nonetheless, the documentation and tutorials are quite good) as well as life hacks with NumPy, for instance, when computing T x f, it happened that the dimensions of T were (x,) and the dimensions of the flux f were (x,y,z); therefore it was impossible to broadcast the operation, but a single trick solved it magically:

T x f = T*f.T → here the dimensions allow for broadcasting
T x f = (T x f).T
(.T stands for transposing in NumPy)

Another difficulty was the volume of the data. When doing the numerical operations required for the task, the laptop froze. To avoid that, I performed the computation only on half of the data, but the same happened, even for one-third of the data, so I decided to do it on one-ninth of the data. I chose these numbers because of the number of slices in the data cube happened to be divisible by 9. Accounting for that, I had to add a test variable to the constructor of Cube_handler so when it is true, the object I'm instancing, loads only one-ninth of the data. I also added a variable indicating which set of slices is being selected, for instance, if this variable is 3, we will be working with the third one-ninth of the slices. Because of that, I tried to add a loop in the implementation of the script to generate 9 images, one for each set of slices, nonetheless, my computer power wasn't able to handle it. In the terminal, I got the message: "process stopped". Therefore I had to do it by hand.

Results

The wavelength ranges for the slices I studied are:

1. 474.86 nm to 525.86 nm
2. 525.98 nm to 576.98 nm
3. 577.11 nm to 628.11 nm
4. 628.23 nm to 679.23 nm
5. 679.36 nm to 730.36 nm
6. 730.48 nm to 781.48 nm
7. 781.61 nm to 832.61 nm
8. 832.73 nm to 883.73 nm
9. 883.86 nm to 934.86 nm

I attach the images generated with the plot.py script for each slice and the name of the filter. The name of the image will follow the next format: image_slicenumber.png, and each of then is contained in a folder with the respective name of the filter.

r_prime:
The wavelength range for this filter is from 530 nm to 730 nm, which matches with the subsets 2,3,4 and 5 from the slices of the data cube. The images seem pretty homogeneous, there is only a variation of one order of

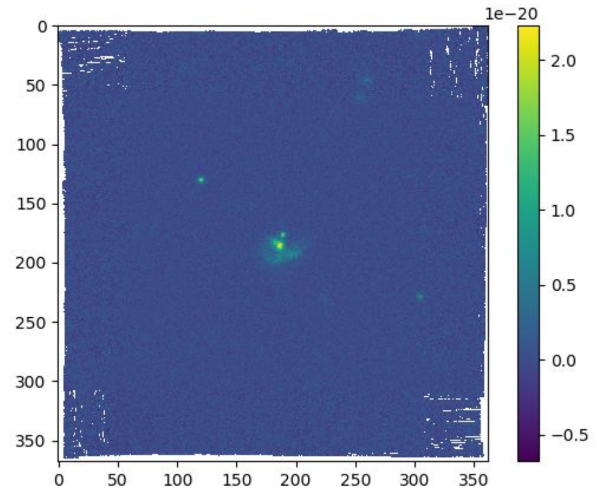
magnitude for the scale of the photometric flux among them. Nothing else can be said. See the attached images.

i_prime:

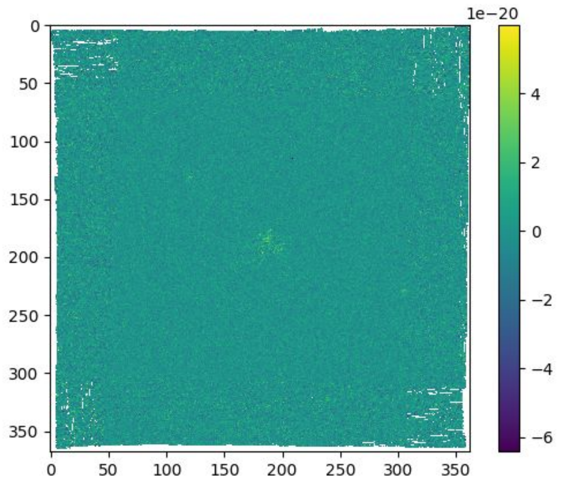
The wavelength range for this filter is from 660 nm to 880 nm, which matches with the subsets 5,6,7 and 8 from the slices of the data cube. The images seem pretty homogeneous, there is only a variation of one order of magnitude for the scale of the photometric flux among them. Nothing else can be said. See the attached images.

674 nm to 681 nm:

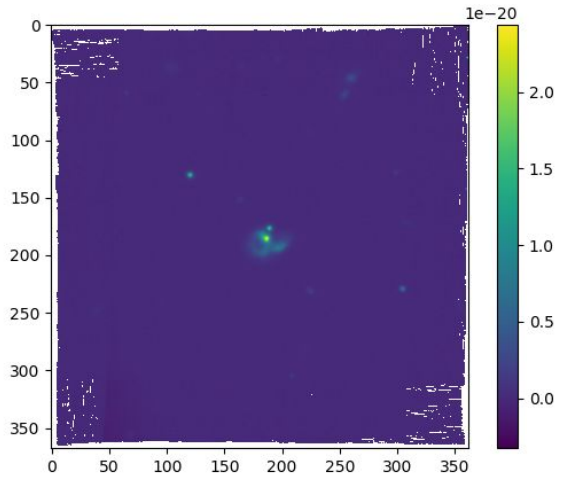
This range matches partially with the subset number 5 of the slices of the data cube. The image of subsets 4 shows the presence of 7 objects, but the S/N is poor for the two on the upper right and the closest to the two central sources:



The image of subsets 5 shows the presence of 1 object in the center, but the S/N is really poor:



683 nm to 689 nm:
This range matches partially with the subset number 5 of the slices of the data cube. Image generated for this the set 5 of slices shows clearly the presence of seven sources, with a good S/N ratio.



Conclusions

From the results, we can see that the broader the range of the filter, the smaller the S/N ratio (i_prime, r_prime), while the smaller the range of the filter (the last two cases), the bigger the S/N ratio. And it is obvious because the broader the range for the filter more light of different wavelengths passes through it, not to mention that we are mainly in the visible (~ 380-740 nm) in the filters for this science case.

Future work

The script can be easily modified such it gets as an argument the number of the set of slices of the data cube upon which it is computing the photometric data, using the **sys** module. If your computer power allows you, you can implement a loop on the code described in the subsection "**The implementation of the algorithm is"** for the nine sets of slices.