Cómo hacer una reducción básica de imágenes FITS con Python

Ricardo Gil-Hutton

Noviembre 2016

En este tutorial vamos a detallar cómo hacer una reducción básica de imágenes astronómicas en formato FITS utilizando Python. El objetivo será corregir las imágenes por todas las contribuciones aditivas y multiplicativas que aparecen y vamos a efectuar el proceso en forma manual sin utilizar scripts para que se comprenda el mecanismo.

Para comenzar hay que copiar las imágenes del objeto, los bias y los flats a un subdirectorio que será nuestro directorio de trabajo. Nunca trabaje sobre las imágenes originales porque si se produce algún error serán necesarias para reiniciar el proceso. En el caso de necesitar darks los mismos se procesaran como una contribución aditiva que depende del tiempo de exposición empleado.

Para comenzar ingresamos a IPython, cargamos los paquetes que vamos a utilizar, cambiamos al directorio de trabajo y hacemos una lista con los nombres de todos los archivos FITS. Recuerde que en IPython es posible guardar en un archivo todos los comandos ingresados utilizando el comando mágico *%logstart -o archivo* al inicio de la sesión y *%logstop* al final, lo que facilitará luego crear un script en base a los comandos ejecutados. Entonces:

```
rgh@Exidor ~> ipython
Python 2.7.9 (default, Mar  1 2015, 12:57:24)
Type "copyright", "credits" or "license" for more information.

IPython 2.3.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: import matplotlib.pyplot as plt

In [2]: import numpy as np

In [3]: import pyfits as ft

In [4]: import glob

In [5]: %cd /home/rgh/program/python/imagenes
/home/rgh/program/python/imagenes

In [6]: lista= glob.glob("*.fit")
```

```
In [7]: print(lista)
['48780001.fit', '48780005.fit', '48780009.fit', '48780013.fit',
'48780017.fit', 'bias0001.fit', 'bias0002.fit', 'bias0003.fit',
'bias0004.fit', 'bias0005.fit', 'bias0006.fit', 'bias0007.fit',
'bias0008.fit', 'bias0009.fit', 'bias0010.fit', 'dflat0001.fit',
'dflat0002.fit', 'dflat0003.fit', 'dflat0004.fit', 'dflat0005.fit',
'dflat0006.fit', 'dflat0007.fit', 'dflat0008.fit', 'dflat0009.fit',
'dflat0010.fit']
```

El paquete **glob** permite hacer listas de archivos en cualquier subdirectorio y expandir los comodines "*" y "?". Utilizando esta lista podemos ver qué tipo de imágenes tenemos utilizando la función *ft.getval()* que retorna el contenido de un *keyword* de la cabecera de una imagen sin necesidad de abrirla con la función *ft.open()*:

```
In [8]: for ii in lista:
   ...:     print("{:} es de tipo {:}".format(ii,ft.getval(ii,'imagetyp')))
   ...:
48780001.fit es de tipo object
48780005.fit es de tipo object
48780009.fit es de tipo object
48780013.fit es de tipo object
48780017.fit es de tipo object
bias0001.fit es de tipo zero
bias0002.fit es de tipo zero
...
...
bias0010.fit es de tipo zero
dflat0001.fit es de tipo flat
dflat0002.fit es de tipo flat
...
...
dflat0010.fit es de tipo flat
```

1 Corrección por overscan:

El **overscan** es una contribución aditiva producida por variaciones electrónicas en el amplificador de lectura del chip. Esta contribución depende estado de la electrónica del equipo en cada instante y puede afectar a cada imágen en forma diferente. Se corrige obligando al amplificador a leer algunas columnas o filas más de las realmente existentes en el chip para poder muestrear la variación temporal de la señal. No todos los equipos permiten leer mas elementos de los que existen físicamente, por lo que no todos los equipos permiten aplicar esta corrección.

Para corregir por overscan lo que se hace es encontrar la región con las columnas o filas adicionales, colapsarlas a una sola dimensión, aplicar algún método para suavizar las variaciones pixel a pixel y restarla a toda la imagen.

Para identificar la región de overscan hay diferentes posibilidades pero para no tener que leer todas las imagenes utilizaremos nuevamente la función *ft.getval()* para verificar los keywords *NAXIS1* y *NAXIS2* ue nos dan las dimensiones de la imagen:

```
In [9]: for ii in lista:
   ...:     print("{:}: {:}x{:}".format(ii,ft.getval(ii,'naxis2'),ft.getval(ii,'naxis1')))
   ...:
48780001.fit: 686x682
48780005.fit: 686x682
48780009.fit: 686x682
48780013.fit: 686x682
48780017.fit: 686x682
bias0001.fit: 686x682
bias0002.fit: 686x682
...
...
bias0010.fit: 686x682
dflat0001.fit: 686x682
dflat0002.fit: 686x682
...
...
dflat0010.fit: 686x682
```

Nótese que invertimos el orden de los ejes para que coincida con lo usual en Python (primero el eje Y, luego el X). Como el chip utilizado es cuadrado, vemos que hay 4 filas adicionales que corresponden a la sección de overscan y es la zona que debemos utilizar.

Como esta corrección hay que aplicarla a todas las imágenes lo mejor es utilizar un bucle *for...in...:* al igual que como se obtuvieron anteriormente las dimensiones y el tipo de imagen, pero primero vamos a aplicar todo el proceso a una sola imagen de prueba para que se comprenda que es lo que se hace. Entonces, primero vamos a leer una imagen cualquiera para usarla de prueba, luego se extraerá la zona de overscan y se la colapsará a una dimensión en la dirección en la cual se encuentran la información adicional calculando su valor medio con la función *np.mean()*:

```
In [10]: ff=ft.open('48780001.fit',uint=True)

In [11]: img=ff[0].data

In [12]: hdr=ff[0].header

In [13]: ff.close()

In [14]: img.shape
Out[14]: (686, 682)

In [15]: sec=img[682:686,:]

In [16]: sec.shape
Out[16]: (4, 682)
```

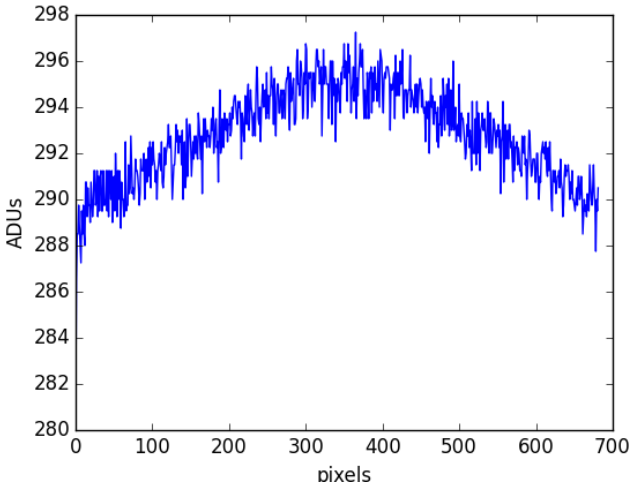

Figure 1:

```
In [17]: ooo=np.mean(sec,axis=0)

In [18]: ooo.shape
Out[18]: (682,)
```

La función *np.mean()* devuelve el valor medio, pero el keyword *axis=0* indica que el promediando solo lo hace a lo largo del primer eje (en este caso el eje Y). Para ver el resultado vamos a graficarlo (ver figura 1):

```
In [19]: plt.figure(1,figsize=(6,4.5)
Out[19]: <matplotlib.figure.Figure at 0x7fa7ef5cfe50>

In [20]: plt.plot(ooo)
Out[20]: [<matplotlib.lines.Line2D at 0x7fa7e1a4bd10>]

In [21]: plt.xlabel("pixels")
Out[21]: <matplotlib.text.Text at 0x7fa7ef3e2290>

In [22]: plt.ylabel("ADUs")
Out[22]: <matplotlib.text.Text at 0x7fa7e1a6e150>
```

En la figura 1 se observa que el ruido es importante debido a los bajos valores de señal por lo cual debemos suavizar los valores para poder proceder a la corrección. Hay diferentes procedimientos para realizar esta tarea siendo el más usual el ajuste de un polinomio de cierto grado. Nosotros aquí vamos a ajustar un polinomio de cuarto grado. Para hacer el ajuste debemos cargar la librería **polynomial** de **numpy**, obtener los coeficientes del polinomio con la función *polyfit()*, y encontrar los valores del polinomio para cada pixel con la función *polyval()*:

```
In [23]: import numpy.polynomial.polynomial as poly
```

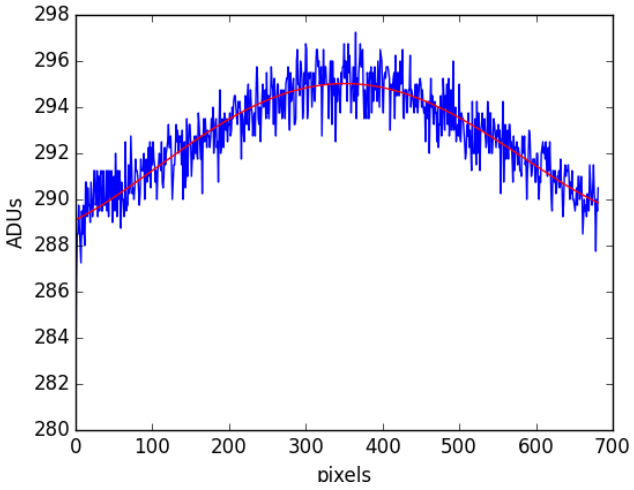

Figure 2:

```
In [24]: xx=np.arange(682)

In [25]: coef=poly.polyfit(xx,ooo,4)

In [26]: print(coef)
[  2.89118555e+02   1.64171511e-02   7.59169786e-05  -2.85842802e-07
   2.07480700e-10]

In [27]: ovs=poly.polyval(xx,coef)

In [28]: plt.plot(ovs,'r-')
Out[28]: [<matplotlib.lines.Line2D at 0x7fa7e19b3390>]
```

En la figura 2 se ve que el ajuste con el polinomio ha reducido considerablemente la dispersión de los valores, por lo que usaremos este vector para el overscan.

Una vez conocidos los valores del overscan se pueden restar a la imagen para corregir por esta contribución y recortar la imagen a su formato correcto sin la zona de overscan que ya no es útil. Para ello, creamos una imagen auxiliar formada por el array *ovs* repetido para toda la imagen, restamos esta imagen a la original, y finalmente la recortamos:

```
In [29]: sca=np.tile(ovs,686).reshape(686,682)

In [30]: img=img-sca

In [31]: img.shape
Out[31]: (686, 682)

In [32]: img=img[0:682,:]
```

```
In [33]: img.shape
Out[33]: (682, 682)
```

La primera instrucción lo que hace es repetir el array *ovs* con el ajuste del overscan 686 veces uno a continuación del otro con la función *np.tile()*, para luego redimensionar al formato correcto con *reshape()*.Si la región de overscan está en la otra dirección (en ese caso tendría una longitud de 686 pixels y habría que repetirlo 682 veces) la primera instrucción sería:

```
In [29]: sca=np.tile(ovs,682).reshape(682,686).transpose()
```

Luego de conseguir la imagen corregida y recortada se debe guardar nuevamente en el disco agregando algún comentario que informe de proceso efectuado. Como esto hay que repetirlo para cada imagen, a continuación se muestra cómo hacer el bucle completo para procesar todas las imágenes, incluyendo la escritura a disco final (aquí se asume que ya se importó el paquete **polynomial**):

```
In [34]: for ii in lista:
    ...:     ff=ft.open(ii,uint=True)
    ...:     img=ff[0].data
    ...:     hdr=ff[0].header
    ...:     ff.close()
    ...:     sec=img[682:686,:]
    ...:     ooo=np.mean(sec,axis=0)
    ...:     xx=np.arange(682)
    ...:     coef=poly.polyfit(xx,ooo,4)
    ...:     ovs=poly.polyval(xx,coef)
    ...:     sca=np.tile(ovs,686).reshape(686,682)
    ...:     img=img-sca
    ...:     img=img[0:682,:]
    ...:     hdr.add_comment("Procesado por OVERSCAN con Python")
    ...:     ft.writeto(ii,img,header=hdr,clobber=True)
```

La cabecera original guardada en *hdr* es modificada automáticamente al escribir a disco para ajustar las dimensiones a los valores reales de la imagen (en este caso, *NAXIS2* que es modificado porque se recortó la imagen). Por otra parte, el keyword *clobber=True* en la función *ft.writeto()* permite sobreescribir el archivo con la imagen corregida.

2 Corrección por bias:

Un listado de las imágenes disponibles para el procesamiento como el que hicimos al principio del tutorial nos indica que tenemos 10 bias para combinar con el objeto de lograr un bias final con buena relación S/N. Si bien el procedimiento más simple es leer todos los bias y obtener diréctamente la media no es recomendable hacer algo así porque no sería posible discriminar los pixels afectados por rayos cósmicos u otros defectos. Por lo tanto, debemos implementar un procedimiento distinto que nos permita aplicar algoritmos para discriminar pixels afectados y usar estimadores estadísticos diferentes a la media en caso de ser necesario.

Lo que haremos a continuación es armar un cubo de datos (un array de tres dimensiones) donde primero se guardan los bias, luego se ordena cada pixel de menor a mayor y por último se obtiene la imagen final combinada aplicando algún algoritmo de discriminación.

Como disponemos de 10 bias de 682 x 682 pixels, primero vamos a armar un cubo de 10 x 682 x 682 donde copiaremos todas las imágenes bias:

```
In [35]: cubo=np.zeros(10,682,682,dtype=float)

In [36]: nro=0

In [37]: for ii in lista:
    ...:     if(ft.getval(ii,'imagetyp') == 'zero'):
    ...:     ff=ft.open(ii)
    ...:     img=ff[0].data
    ...:     hdr=ff[0].header
    ...:     ff.close()
    ...:     cubo[nro,:,:]=np.copy(img)
    ...:     nro+=1
```

Ahora ordenaremos el cubo de manera que cada pixel esté ordenado de mayor a menor utilizando la función *np.sort()* aplicada a lo largo del primer eje:

```
In [38]: cubo=np.sort(cubo,axis=0)
```

Supongamos ahora que queremos combinar las imágenes para obtener el valor medio despreciando el valor más alto de cada pixel para evitar una posible contaminación por rayos cósmicos, entonces calculamos la media a lo largo del primer eje utilizando las primeras nueve imágenes bias:

```
In [39]: zero=np.mean(cubo[0:9,:,:],axis=0)

In [40]: zero.shape
Out[40]: (682, 682)
```

Si, por ejemplo, en lugar del valor medio se quiere usar la mediana y discriminar el valor más alto hay que ejecutar:

```
In [39]: zero=np.median(cubo[0:9,:,:],axis=0)
```

Como ahora tenemos un bias de buena relación S/N podemos usarlo para restar a las imágenes de objeto y flats, guardando cada imagen nuevamente a disco:

```
In [41]: for ii in lista:
    ...:     if(ft.getval(ii,'imagetyp') == 'object'):
    ...:     ff=ft.open(ii)
    ...:     img=ff[0].data
    ...:     hdr=ff[0].header
    ...:     ff.close()
```

```
    ...:     img=img-zero
    ...:     hdr.add_comment("Procesado por BIAS con Python")
    ...:     ft.writeto(ii,img,header=hdr,clobber=True)

In [42]: for ii in lista:
    ...:     if(ft.getval(ii,'imagetyp') == 'flat'):
    ...:     ff=ft.open(ii)
    ...:     img=ff[0].data
    ...:     hdr=ff[0].header
    ...:     ff.close()
    ...:     img=img-zero
    ...:     hdr.add_comment("Procesado por BIAS con Python")
    ...:     ft.writeto(ii,img,header=hdr,clobber=True)
```

3 Corrección por flat:

Ya sabemos que tenemos 10 imágenes flat para combinar y vamos a utilizar un procedimiento similar al usado para combinar los bias, pero como es posible que los niveles de iluminación de cada flat sean diferentes en este caso vamos a darle peso a cada imagen con su mediana:

```
In [43]: cubo=np.zeros(10,682,682,dtype=float)

In [44]: nro=0

In [45]: sum=0.

In [46]: for ii in lista:
    ...:     if(ft.getval(ii,'imagetyp') == 'flat'):
    ...:     ff=ft.open(ii)
    ...:     img=ff[0].data
    ...:     hdr=ff[0].header
    ...:     ff.close()
    ...:     med=np.median(img)
    ...:     sum+=med
    ...:     cubo[nro,:,:]=np.copy(img)*med
    ...:     nro+=1

In [47]: flat=np.mean(cubo[0:9,:,:],axis=0)/sum
```

Como ahora tenemos un flat de buena relación S/N podemos usarlo para hacer la corrección de las imágenes de objeto dividiendo por el flat y rescalando con su media para no modificar el valor medio de cada imagen:

```
In [48]: mflt=np.mean(flat)

In [49]: for ii in lista:
```

```
...:     if(ft.getval(ii,'imagetyp') == 'object'):
...:     ff=ft.open(ii)
...:     img=ff[0].data
...:     hdr=ff[0].header
...:     ff.close()
...:     img=img/flat*mflt
...:     hdr.add_comment("Procesado por FLAT con Python")
...:     ft.writeto(ii,img,header=hdr,clobber=True)
```

4 Corrección por dark:

Finalmente, en el caso que se necesite corregir por dark el proceso para combinarlos es similar al utilizado para los bias ya que es una contribución aditiva. En el caso de los darks hay que hacer la salvedad de que habrá que escalarlos por tiempo de exposición dividiendo primero por el tiempo utilizado para los darks para luego multiplicar por el tiempo empleado en la imagen a corregir (usualmente diferente en el caso de objetos y flats). Lo más práctico siempre es adquirir darks con un tiempo de exposición igual al utilizado para las imágenes a corregir lo que permite en este caso combinar en una sola imagen las contribuciones aditivas de bias y dark por lo cual se aplican ambas correcciones de una vez.