Cómo trabajar con imágenes FITS en Python

Ricardo Gil-Hutton

Diciembre 2019

1 Operando con imágenes FITS

Para operar con imágenes FITS existe en el repositorio de Python un módulo denominado **pyfits** que permite leer, escribir y trabajar con este formato de imagen en una, dos o varias dimensiones. Esta sección muestra una rápida introducción a **pyfits** que permitirá que un usuario nuevo pueda trabajar sin problemas en Python.

Para instalar este módulo se puede utilizar el programa **pip3** (disponible en los repositorios de paquetes de Linux como python3-pip). Desde una terminal y como root hacer:

```
root@Exidor: > pip3 install pyfits
```

Salir del root, iniciar el intérprete de Python o IPython, e importar numpy y pyfits. Yo tengo seteado IPython para que al inicio importe automáticamente varios módulos, pero es posible hacerlo manualmente ejecutando:

```
In [1]: import numpy as np

In [2]: import pyfits as ft
```

Otra opción posible, y tal vez la más práctica, es utilizar la versión de **pyfits** que se encuentra disponible en el paquete **astropy** ya que es posible que quien trabaje con imágenes FITS necesite cargar los módulos que se encuentran disponibles en este paquete. En este caso, y asumiendo que se tiene instalado **astropy**, el módulo se importa con:

```
In [2]: import astropy.io.fits as ft
```

Una vez cargado el módulo es puede abrir una imagen mediante:

```
In [3]: ff=ft.open('xxx.fits',uint=True)
```

La función *open* tiene varios parámetros que permiten abrir el archivo FITS de diferentes maneras. Aquí utilizamos el parámetro *uint=True* para asegurar que está leyendo enteros sin signo. La función devuelve una lista de objetos tipo HDU ("Header Data Unit") que identifican las cabeceras / imágenes guardadas en el archivo. Por ejemplo, la primera imagen en la lista *ff* es ff[0]; si existiera una segunda sería ff[1], etc. Para obtener información sobre el archivo se hace:

```
In [4]: ff.info()
Filename: xxx.fits
No.    Name         Type      Cards   Dimensions   Format
0    PRIMARY     PrimaryHDU      71   (512, 512)   int16
```

que nos dice que hay sólo una imagen guardada en el archivo (identificada con *0*) y que tiene dimensiones de 512×512 pixels.

Cuando se termine de operar con el archivo FITS es conveniente cerrarlo con:

```
In [5]: ff.close()
```

1.1 Leyendo la cabecera:

Cada objeto del listado que devuelve *open* es un objeto HDU que contiene una cabecera y datos. La cabecera es un objeto especial de **pyfits** que contiene los keywords con una estructura parecida a la de un dictionary. Es posible leerla a una variable haciendo:

```
In [6]: hdr=ff[0].header

In [7]: type(hdr)
Out[7]: pyfits.header.Header

In [8]: hdr
Out[8]:
SIMPLE  =                    T / Fits standard
BITPIX  =                   16 / Bits per pixel
NAXIS   =                    2 / Number of axes
NAXIS1  =                  512 / Axis length
NAXIS2  =                  512 / Axis length
EXTEND  =                    F / File may contain extensions
ORIGIN  = 'NOAO-IRAF FITS Image Kernel July 2003' / FITS file originator
DATE    = '2004-07-20T20:35:38' / Date FITS file was generated
IRAF-TLM= '17:35:38 (20/07/2004)' / Time of last modification
OBJECT  = 'm51  B  600s'       / Name of the object observed
IRAF-MAX=           1.993600E4  /  DATA MAX
IRAF-MIN=          -1.000000E0  /  DATA MIN
CCDPICNO=                   53  /  ORIGINAL CCD PICTURE NUMBER
ITIME   =                  600  /  REQUESTED INTEGRATION TIME (SECS)
TTIME   =                  600  /  TOTAL ELAPSED TIME (SECS)
OTIME   =                  600  /  ACTUAL INTEGRATION TIME (SECS)
DATA-TYP= 'OBJECT (0)'          /  OBJECT,DARK,BIAS,ETC.
DATE-OBS= '05/04/87'            /  DATE DD/MM/YY
RA      = '13:29:24.00'         /  RIGHT ASCENSION
DEC     = '47:15:34.00'         /  DECLINATION
EPOCH   =                0.00  /  EPOCH OF RA AND DEC
ZD      = '22:14:00.00'         /  ZENITH DISTANCE
```

```
UT      = ' 9:27:27.00'         /  UNIVERSAL TIME
ST      = '14:53:42.00'         /  SIDEREAL TIME
CAM-ID  =                    1  /  CAMERA HEAD ID
CAM-TEMP=              -106.22  /  CAMERA TEMPERATURE, DEG C
DEW-TEMP=              -180.95  /  DEWAR TEMPRATURE, DEG C
F1POS   =                    2  /  FILTER BOLT I POSITION
F2POS   =                    0  /  FILTER BOLT II POSITION
TVFILT  =                    0  /  TV FILTER
CMP-LAMP=                    0  /  COMPARISON LAMP
TILT-POS=                    0  /  TILT POSITION
...
...
...
```

pero también se puede leer directamente el contenido de un keyword:

```
In [8]: ff[0].header['DATE-OBS']
Out[8]: '05/04/87'

In [9]: hdr['DATE-OBS']
Out[9]: '05/04/87'
```

o de una línea específica de la cabecera:

```
In [10]: ff[0].header[9]
Out[10]: 'm51  B  600s'

In [11]: hdr[9]
Out[11]: 'm51  B  600s'
```

Si se desea reemplazar el valor de algún keyword:

```
In [12]: hdr['object']='Messier 51'

In [13]: hdr[9]
Out[13]: 'Messier 51'
```

Si el keyword no existe se lo puede agregar facilmente haciendo:

```
In [10]: hdr['Lugar']='San Juan'
```

Si se quiere agregar un comentario o un keyword *history* a la cabecera existen métodos especiales:

```
In [11]: hdr.add_history('Se modifico hoy')

In [12]: hdr.add_comment('Es facil de operar con la cabecera')
```

Hay muchos otros métodos disponibles que no tratamos en este tutorial pero que se pueden investigar utilizando el help. Por ejemplo, si queremos leer, modificar o agregar un keyword a la cabecera de un archivo FITS sin leer el archivo completo desde el disco se pueden utilizar las funciones *ft.getval* y *ft.setval*.

1.2 Leyendo los datos:

Para leer los datos se utiliza un método similar al que usamos para la cabecera pero ahora el objeto resultante es un array de numpy y se pueden usar todos los métodos disponibles para operar y manejar arrays:

```
In [13]: img=ff[0].data

In [14]: type(img)
Out[14]: numpy.ndarray

In [15]: img.shape
Out[15]: (512,512)
```

Hay que prestar atención a dos detalles importantes: primero, la imagen guardada en un archivo FITS sigue una norma standard fijada para Fortran y que se respeta en IRAF que define el incio del array en el pixel (1,1) pero en Python los arrays se inician en la celda (0,0), **por lo cual hay que prestar atención a este desfasaje de 1 pixel en ambas coordenadas cuando se hagan cálculos o se necesite referir la posición de un objeto en coordenadas de la imagen**. Segundo, en los arrays de Python el eje que más rápido cambia es el eje Y asi que el valor del pixel que se encuentra en la posición X=5,Y=10 se obtiene haciendo:

```
In [16]: a=img[10,5]
```

Por esta razón, la información que se encuentra en la cabecera de la imagen debe considerarse en este orden. Por ejemplo, en la cabecera de la imagen aparecen los keywords *NAXIS1*, *NAXIS2*, etc. que corresponden tradicionalmente al eje X, Y, etc., siendo el eje X el que más rápido varía, asi que si quiero crear un array similar al *img* con las dimensiones correctas debo hacer:

```
In [17]: xx=hdr['naxis1']

In [18]: yy=hdr['naxis2']

In [19]: img1=np.zeros((naxis2,naxis1))
```

2 Desplegando imágenes FITS

El procedimiento para ver una imagen FITS en Python es muy simple: sólo es necesario utilizar las funciones y métodos disponibles en **matplotlib.pyplot**:

```
In [20]: import matplotlib.pyplot as plt

In [21]: plt.figure(1)

In [22]: plt.imshow(img,cmap='Greys',vmin=0,vmax=200)
Out[22]: <matplotlib.image.AxesImage at 0x7f759aed4e10>
```

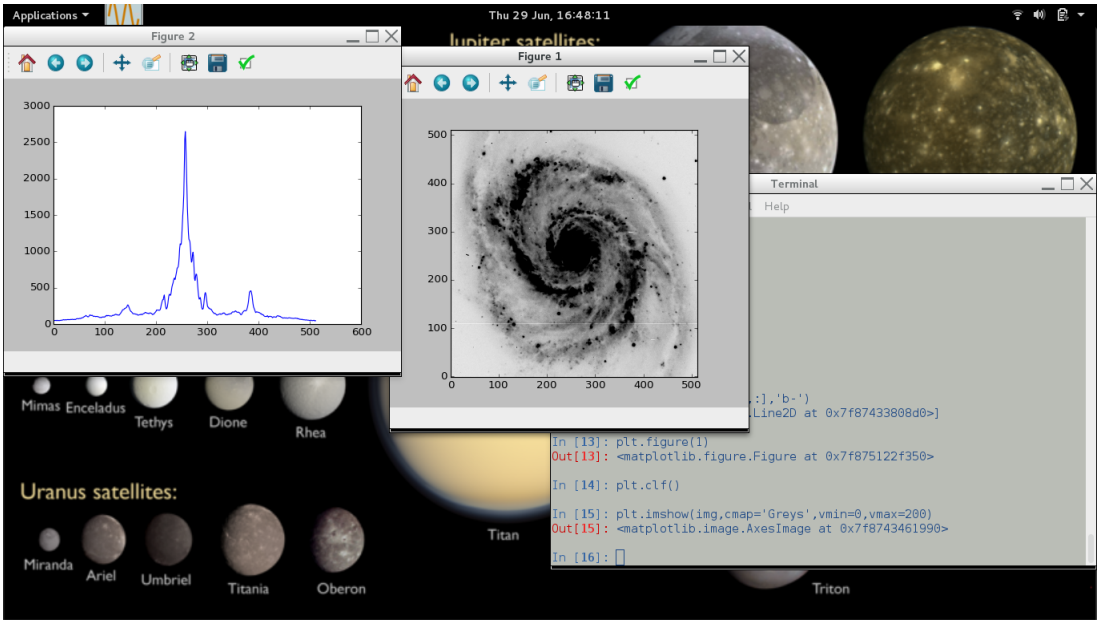

Figure 1:

donde el parámetro *cmap* indica el mapa de colores a utilizar en la paleta y *vmin* y *vmax* el valor mínimo y máximo asignados a los extremos de la paleta. Si se quiere, se puede abrir otra ventana para desplegar otra imagen, hacer un gráfico, etc. Por ejemplo, un corte por el centro de la imagen sería:

```
In [23]: x=range(512)

In [24]: plt.figure(2)

In [25]: plt.plot(x,img[256,:],'r-')
Out[25]: [<matplotlib.lines.Line2D at 0x7f759a8a8490>]
```

Hay muchos comando útiles de **matplotlib.pyplot** que pueden usarse para hacer diferentes cosas pero su explicación escapa a este tutorial. Si es importante destacar que si uno quiere usar una ventana para desplegar una imagen nueva o hacer un gráfico es necesario borrar antes el contenido con *plt.clf()* porque sino lo sobreescribirá. Si se quiere cerrar una ventana abierta para desplegar una imagen o gráfico se debe utilizar el comando *plt.close()*. Es recomendable no tener demasiadas ventanas abiertas para evitar que el intérprete Python se haga más lento.

3 Usando SAOImage ds9 para desplegar imágenes

Si bien es posible en Python desplegar imágenes utilizando **matplotlib.pyplot** y la función *imshow()*, en general esto resulta algo molesto porque usualmente se requiere hacer zoom, cambiar paletas, determinar centros de objetos y otros procesos rutinarios en el procesamiento de imágenes para el cual la función *figure* no está bien preparada.

Una opción interesante es utilizar **SAOImage ds9** como visor de imágenes ya que es posible controlar ds9 mediante mensajes XPA que es el mismo sistema que utiliza IRAF para interactuar con este programa. Aquí se darán las instrucciones necesarias para bajar el paquete y un breve instructivo de como operar desde Python.

3.1 El módulo Pyds9:

En los repositorios de Python se ofrecen varios módulos que permiten lograr la conexión mediante mensajes XPA entre Python y ds9. Entre ellos podemos mencionar **numdisplay** creado por el Space Telescope Science Institute (http://stsdas.stsci.edu/numdisplay/), o **pysao** que se ofrece en los repositorios de Python (https://pypi.python.org/pypi/pysao), pero personalmente prefiero otro módulo denominado **pyds9** que me resulta mucho más versátil y práctico y se obtiene en la página del ds9 (http://hea-www.harvard.edu/RD/pyds9/).

Para instalar *pyds9* se puede utilizar el programa **pip3** (disponible en los repositorios de paquetes de Linux como python3-pip). Desde una terminal y como root hacer:

```
root@Exidor: > pip3 install pyds9
```

Salir del root, iniciar el intérprete de Python, e importar Numpy, Pyfits y Pyds9. Yo tengo seteado IPython para que al inicio importe varios módulos automáticamente, pero es posible hacerlo manualmente ejecutando:

```
In [1]: import numpy as np

In [2]: import pyfits as ft

In [3]: import pyds9 as sao
```

Supongamos que tenemos en disco una imagen FITS llamada *xxx.fits* y que queremos trabajar en ella con Python y ds9. El primer paso es cargarla en un array:

```
In [4]: ff=ft.open('xxx.fits',uint=True)

In [5]: img=ff[0].data

In [6]: hdr=ff[0].header
```

Ahora ya tenemos la imagen en el array *img* y su cabecera en *hdr*. A continuación debemos desplegar ds9 y mostrar la imagen:

```
In [7]: ds9=sao.DS9('/opt/iraf-v2.16/ds9-7.2')

In [8]: ds9.set_np2arr(img,dtype=np.int16)
```

La instrucción 7 carga una versión específica de ds9 (la 7.2 que tiene el nombre "ds9-7.2"), pero si no se especifica una versión en particular se cargará la que está en el path. En la instrucción 8 se despliega la imagen en ds9 pero como este programa tiene algunas limitaciones respecto al tipo de datos que se puede manejar se convierte primero la imagen a un entero de 16 bits. Es claro que la imagen que acabamos de desplegar es muy oscura debido a que los valores no han sido escalados como

corresponde, pero la ventaja de este módulo es que se puede enviar cualquier tipo de comando XPA al ds9 para ejecutar diferentes tareas. Por ejemplo, si se clickea en el menú del ds9 en la pestaña "Scale" se puede ver que el escalado ha sido lineal y considerando el máximo y el mínimo de la imagen para desplegar la paleta. Si queremos modificar la escala y la paleta desde Python podemos utilizarlos botones o el menú del ds9, o hacer desde IPython:

```
In [9]: ds9.set("scale zscale")

In [10]: ds9.set("cmap Heat")
```

donde "scale zscale" o "cmap Heat" son comandos XPA, que usualmente coinciden con las posibilidades que aparecen en las pestañas del menú de ds9. Si queremos averiguar las coordenadas de un objeto en la imagen se puede utilizar:

```
In [11]: ds9.get("imexam key coordinate image")
Out[11]: 'space  442 410'
```

donde Python se queda esperando a que se posicione el cursor sobre la imagen y se presione una tecla, en este caso la barra espaciadora, para devolver un string con la tecla y las coordenadas de la imagen según Python. Es importante recordar que ds9 utiliza un sistema de coordenas diferente al de los arrays de Python, asi que las coordenadas que devuelve *ds9.get* son coordenadas de **ds9 en orden (X,Y)** que hay que desfasar en un pixel para que coincidan con el array *img* que tenemos en memoria. Si posicionamos el cursor sobre esas coordenadas en el ds9 vemos que ese pixel tiene un valor de 3164 y en el array:

```
In [12]: img[409,441]
Out[12]: 3164
```

Lo importante es que ds9 es un reemplazo completo para desplegar todo tipo de array, no solo imágenes. Por ejemplo, podemos crear un array cualquiera y mostrarlo en ds9:

```
In [12]: aa=np.arange(400*400).reshape(400,400)

In [13]: ds9.set_np2arr(aa)
```

Un listado completo de los posibles comandos XPA se pueden encontrar en diferentes páginas web (por ejemplo, http://ds9.si.edu/doc/ref/xpa.html), pero en la sección 4 se incluye un resumen de comandos usuales para operar con imágenes en ds9 desde Python.

A manera de ejemplo, a continuación se muestra un script de Python que permite interactuar con ds9 para realizar cortes en ambas coordenadas y mostrar el correspondiente gráfico.

```
def cortexy(imagen,vis,plot):
    """Esta funcion opera con ds9 para elegir un
    punto de la imagen y realizar un corte a lo largo
    de la coordenada X o Y y lo muestra en una figura.

    Uso:

      cortexy(img,ds9,plt)
```

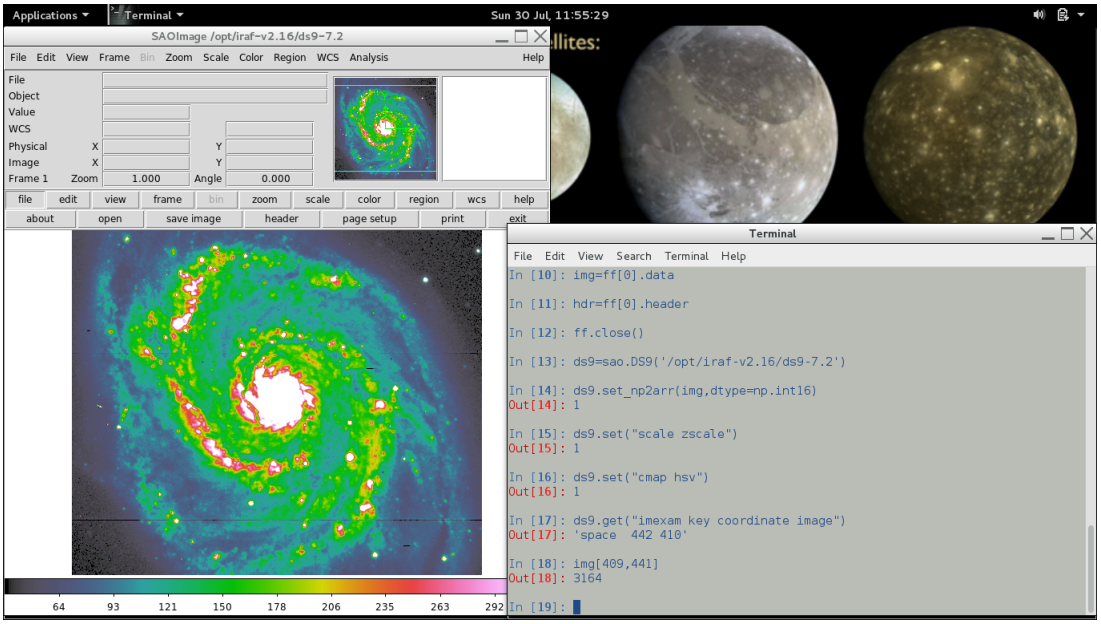

Figure 2:

```
    Parametros:

    imagen    : array que esta desplegado en el ds9
    vis       : objeto creado por Pysao
    plot      : modulo del matplotlib.pyplot

    Retorna:

    la funcion no retorna ningun valor

    """

# se queda esperando hasta que se presione X o Y
# en la imagen
    print("sobre la imagen presione x o y")
    lista=vis.get("imexam key coordinate image")

# crea una lista al descomponer la string que
# devuelve ds9. En cod va la tecla presionada
#
# notese que la string usa codigos unicode
# para poder representar todo tipo de tecla
    ll=lista.split()
    cod=ll[0]
    x=int(ll[1])
    y=int(ll[2])
```

```
# dimensiones de la imagen y valores a
# graficar
    size=imagen.shape
    if(cod == u'x')or(cod == u'X'):
      cox=range(size[1])
      coy=imagen[y-1,:]
    elif(cod == u'y')or(cod == u'Y'):
      cox=range(size[0])
      coy=imagen[:,x-1]
    else:

# si no presiona X o Y retorna sin hacer nada
      return

# hace el grafico
    plot.figure(figsize=(6,4.5))

    if(cod == u'x')or(cod == u'X'):
      plot.title("Corte en X para Y={:d}".format(y-1))
      plot.xlabel("Coord. X")
    else:
      plot.title("Corte en Y para X={:d}".format(x-1))
      plot.xlabel("Coord. Y")

    plot.ylabel("Valor Pixel")
    plot.plot(cox,coy,'b-')

    return
```

Para ejecutarlo copie el script a un archivo, por ejemplo "prueba-ds9.py", y lealo desde IPython con:

```
In [14]: %run prueba-ds9
```

y ejecútelo con:

```
In [15]: cortexy(img,ds9,plt)
```

La función va a esperar que presione la tecla "X" o "Y" sobre la imagen y le mostrará una figura con el correspondiente corte. Es posible modificar el script para que al clickear sobre la imagen haga ambos cortes al mismo tiempo, pero eso queda como ejercicio.

3.2 Algunas opciones más avanzadas de Pyds9:

El módulo **pyds9** también permite operar desde Python con una ventana ds9 que se encuentre abierta previamente. Para eso debemos averiguar el identificador que utiliza XPA para localizar la ventana: entrando en el menú del ds9 que está abierto se clickea en (*file - XPA - Information*) lo que abrirá una nueva ventana con información XPA. El dato que se debe guardar es el indicado como XPA_METHOD que, en el caso mostrado en la figura 4, es *7f000101:39221* correspondiente a *IP:Port*.

Entonces, en la ventana de IPython hacemos:

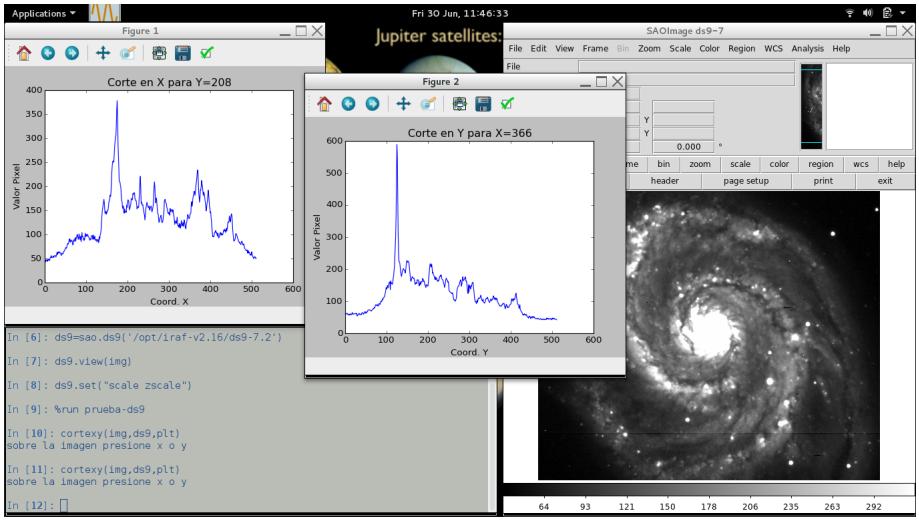

Figure 3:

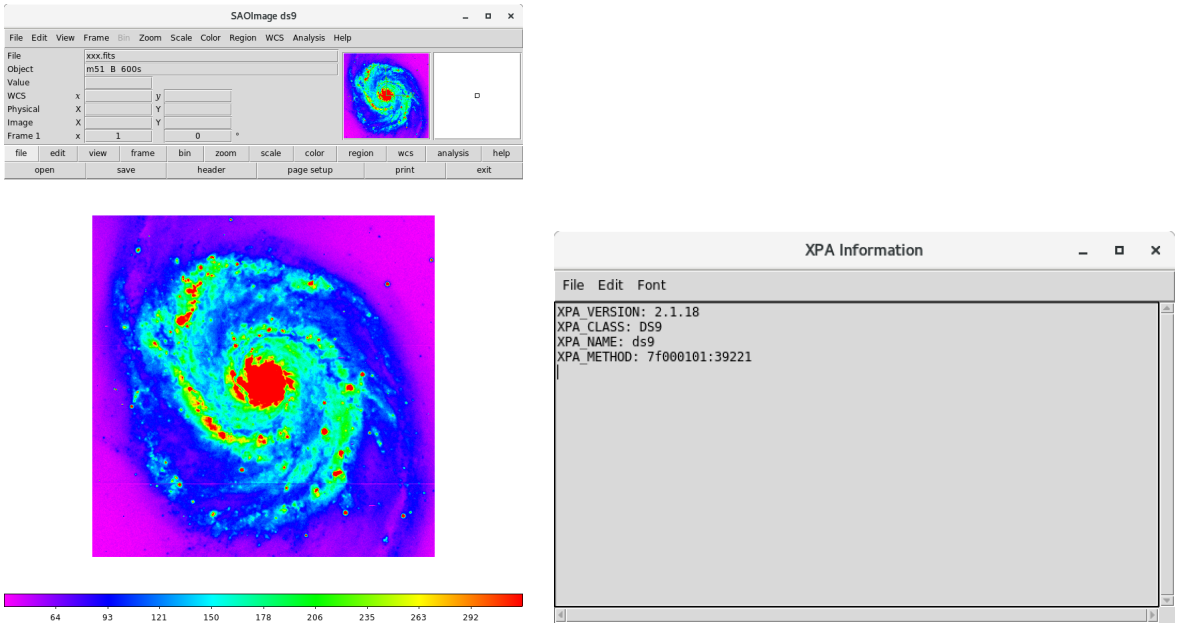

Figure 4:

```
In [16]: ds9c=sao.DS9('7f000101:39221')
```

y el ds9 que estaba abierto pasa a ser una nueva ventana para desplegar imágenes desde Python. Mediante este procedimiento se pueden utilizar tantas ventanas de ds9 como se desee.

Pyds9 también permite transferir a Python una imagen que se encuentra previamente desplegada en alguna de las ventanas del ds9. Para ello es necesario primero identificar el ds9 para operar desde Python siguiendo el procedimiento que se indicó previamente, para luego ejecutar:

```
In [17]: arr=ds9c.get_arr2np()

In [18]: np.shape(arr)
Out[18]: (512, 512)

In [19]: arr
Out[19]:
array([[38, 43, 35, ..., 45, 43, 41],
       [36, 41, 37, ..., 42, 41, 39],
       [38, 45, 37, ..., 42, 35, 43],
       ...,
       [49, 52, 49, ..., 41, 35, 39],
       [57, 52, 49, ..., 40, 41, 43],
       [53, 57, 57, ..., 39, 35, 45]], dtype=int16)
```

que guarda en el array *arr* la imagen que estaba desplegada en ds9.

4 Resumen de comandos usuales para operar ds9 desde Python

```
ds9=sao.DS9() # abre ds9
ds9=sao.DS9('/iraf/ds9-7.2') # abre una version particular del ds9

ds9.set_np2arr(img,dtype=np.int16) # muestra un array
img=ds9.get_arr2np()         # guarda en un array la imagen desplegada en ds9

ds9.set('file test.fits') # muestra directamente una imagen FITS
ds9.get('file') # obtienen info como string si se cargo directamente
ds9.get("fits width")
ds9.get("fits height")
ds9.get("fits depth")
ds9.get("fits size")
ds9.get("fits bitpix")
ds9.get("fits type")
ds9.get("fits header")
ds9.get("fits header 2") # muestra la cabecera de la segunda extension
ds9.get("fits header keyword 'object'")

ds9.get("cmap") # nombre de la paleta de color
ds9.set("cmap Heat")
ds9.set("cmap invert yes")
ds9.set("cmap invert no")

ds9.set("colorbar yes")  # muestra la barra de paleta de colores
ds9.set("colorbar no")
ds9.set("colorbar vertical")
ds9.set("colorbar horizontal")
ds9.set("colorbar numerics yes")
ds9.set("colorbar size 20") # ancho de la barra
ds9.set("colorbar ticks 10")

ds9.set("crosshair 100 100 physical") # posiciona el cursor
ds9.set("cursor 20 20") # mueve el cursor el nro de pixels indicado

ds9.get("data physical 250 250 50 50 yes") # devuelve un bloque de datos desde el vertice
ds9.get("data physical 250 250 50 50 no") # idem al anterior, pero indicando tambien las c
ds9.get("data image 250 250 50 50 yes") # idem pero en coord. de la imagen.

ds9.set("quit") # cierra el ds9
ds9.set("exit") # idem

ds9.set("export png prueba.png") # saca imagen como png
ds9.set("export jpg prueba.jpeg 75") # saca imagen como jpg con cierta calidad
```

```
ds9.get("frame") # indica el frame activo
ds9.get("frame all") # indica todos los frames abiertos
ds9.set("frame 2")  # pasa al frame 2
ds9.set("frame center")
ds9.set("frame clear")
ds9.set("frame new") # crea un nuevo frame
ds9.set("frame delete") # borre el frame actual
ds9.set("frame show 2") # muestra el frame 2
ds9.set("frame first")
ds9.set("frame last")
ds9.set("frame prev")
ds9.set("frame next")

ds9.set("header") # abre la ventana del header
ds9.set("header 2")
ds9.set("header close")
ds9.set("header save 1 hhh.txt") # guarda el header en el archivo hhh.txt

ds9.get("imexam coordinate image") # devuelve coord. donde se presiona mouse
ds9.get("imexam key coordinate image") # devuelve coord. y tecla presionada
ds9.get("imexam any coordinate image") # devuelve coord. y tecla o mouse
ds9.get("imexam data") # devuelve valor de pixel
ds9.get("imexam key data") # devuelve valor de pixel y tecla
ds9.get("imexam any data") # devuelve valor de pixel y tecla o mouse
ds9.get("imexam data 3 3") # devuelve valor de pixels en un cuadro de 3x3

ds9.set("magnifier color yellow") # color del magnifier
ds9.set("magnifier zoom 4") # zoom del magnifier
ds9.set("magnifier cursor no") # sin cursor en el magnifier

ds9.set("minmax mode auto") # fija el modo del minmax automaticamente
ds9.set("minmax mode scan") # fija el modo del minmax revisando la imagen
ds9.set("minmax mode sample") # fija el modo del minmax muestreando
ds9.set("minmax interval 10") # en sample, muestrea cada 10 valores

ds9.set("orient x") # invierte en la direccion indicada
ds9.set("orient y")
ds9.set("orient xy")
ds9.set("orient none")
ds9.set("orient open") # abre la ventana de dialogo sobre orientacion, etc
ds9.set("orient close")

ds9.get("pan") # valor del centro de la imagen sobre el cual se rota, etc.
ds9.set("pan to 100 100 physical") " posiciona nuevo centro
ds9.set("pan 100 100 physical") " posiciona nuevo centro en forma relativa
ds9.set("pan open") # abre la ventana de dialogo sobre orientacion, etc
```

```
ds9.set("pan close")

ds9.set("rotate 45") # rota la imagen 45 grados en forma relativa
ds9.set("rotate to 30") # rota a un angulo de 30 grados
ds9.set("rotate open") # abre la ventana de dialogo sobre orientacion, etc
ds9.set("rotate close")

ds9.set("save fits sale.fits image") # guarda la imagen como FITS
ds9.set("saveimage eps xx.eps") # guarda snapshot como eps
ds9.set("saveimage jpeg xx.jpg 75") # guarda como jpeg
ds9.set("saveimage png xx.png") # guarda como png

ds9.set("scale linear") # controla los limites y escala de colores
ds9.set("scale log 100")
ds9.set("scale histequ")
ds9.set("scale limits 1 100")
ds9.set("scale mode zscale")
ds9.set("scale mode 99.5")
ds9.set("scale open")
ds9.set("scale close") # abre la ventana de dialogo sobre la escala de colores

ds9.set("blink")  # blinkea las imagenes
ds9.set("single") # muestra una sola imagen
ds9.set("tile yes") # muestra varias imagenes
ds9.set("tile row")
ds9.set("tile column")

ds9.set("tile grid layout 2 2")
ds9.set("view layout vertical") # controla que se muestra en la ventana
ds9.set("view layout horizontal")
ds9.set("view info yes")
ds9.set("view panner yes")
ds9.set("view magnifier yes")
ds9.set("view buttons yes")
ds9.set("view colorbar yes")
ds9.set("view graph horizontal yes")
ds9.set("view graph vertical yes")
ds9.set("view filename yes")
ds9.set("view object yes")
ds9.set("view minmax yes")
ds9.set("view lowhigh yes")
ds9.set("view frame yes")

ds9.set("zoom 2") # fija el zoom de manera relativa al doble
ds9.set("zoom to 4") # fija el zoom en 4 de manera absoluta
ds9.set("zoom to fit") # fija el zoom para que ajuste a la ventana
ds9.set("zoom open") # abre la ventana de dialogo sobre orientacion, etc
```

```
ds9.set("zoom close")

ds9.set("regions color blue")
ds9.set("regions command {circle 300 220 20 # color=blue}") # centro y radio OJO: orden X
ds9.set("regions command {box 200 220 80 60 # color=cyan width=3}")
ds9.set("regions command {box 300 220 40 60}") # centro y lados
ds9.set("regions command {text 230 120 #text="Fun with ds9" font="times 18 bold"}")
ds9.set("regions delete all")
ds9.set("regions delete select")
ds9.set("regions load /home/rgh/program/python/ds9.reg")
ds9.set("regions /home/rgh/program/python/ds9.reg")
ds9.set("regions save /home/rgh/program/python/ds9.reg")
```