Escuela de Ciencias Físicas y Matemáticas
Física Aplicada
Luis Fernando Icú Son
201804765

Manual de instalación y uso de PyBDSF

1. Introducción

En el ámbito de la astronomía es necesaria la identificación de las fuentes en un mapa y su distinción del ruido instrumental y el ruido de confusión. Para realizar esta tarea se cuenta con distintos softwares entre los cuales se encuentra el **Python Blob Detector and Source Finder (PyBDSF)**. Esta herramienta es capaz de distinguir las fuentes del ruido y crear un catálogo con las coordenadas de cada una de las fuentes detectadas. La documentación para la descarga y utilización de PyBDSF puede encontrarse aquí.

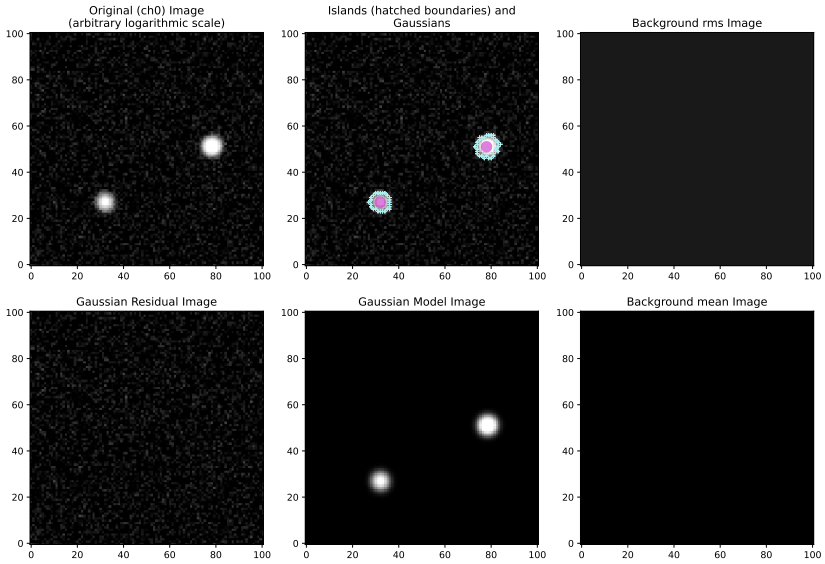

Figura 1: Resultado de PyBDSF.

2. Instalación de PyBDSF

La página mencionada anteriormente menciona algunos métodos de instalación. Aquí voy a mencionar el método que me funcionó a mí.

2.1. Librerías necesarias

Antes de la instalación de PyBDSF es necesaria la instalación de algunas librerías. Para la instalación de ciertas librerías puede ser requerido el uso del superusuario mediante el comando *sudo*. Las librerías que se instalaron fueron:

1. *numpy* en su version 1.22.2.

```
pdmartinez@informatica:~$ python3 -m pip install numpy==1.22.2
Defaulting to user installation because normal site-packages is not writeable
Collecting numpy==1.22.2
  Downloading numpy-1.22.2-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.
whl (16.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 16.8/16.8 MB 30.9 MB/s eta 0:00:00
Installing collected packages: numpy
  Attempting uninstall: numpy
    Found existing installation: numpy 1.24.4
    Uninstalling numpy-1.24.4:
      Successfully uninstalled numpy-1.24.4
Successfully installed numpy-1.22.2
```

2. *astropy*

```
pdmartinez@informatica:~/Documentos/PyBDSF$ pip install astropy
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: astropy in /home/pdmartinez/.local/lib/python3.8/
site-packages (5.2.2)
```

3. *matplotlib* en su version 3.5.0.

```
pdmartinez@informatica:~$ python3 -m pip install matplotlib==3.5.0
Defaulting to user installation because normal site-packages is not writeable
Collecting matplotlib==3.5.0
  Downloading matplotlib-3.5.0-cp38-cp38-manylinux_2_5_x86_64.manylinux1_x86_64.
whl (11.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.3/11.3 MB 31.3 MB/s eta 0:00:00
```

4. *pyfits*.

```
pdmartinez@informatica:~/Documentos/PyBDSF$ pip install pyfits
Defaulting to user installation because normal site-packages is not writeable
Collecting pyfits
  Downloading pyfits-3.5.tar.gz (1.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 4.8 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
```

5. *pywcs*.

```
WARNING: There was an error checking the latest version of pip.
pdmartinez@informatica:~/Documentos/PyBDSF$ pip install pywcs
Defaulting to user installation because normal site-packages is not writeable
Collecting pywcs
  Downloading pywcs-1.12.zip (2.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.4/2.4 MB 8.2 MB/s eta 0:00:00
```

6. *setuptools*.

```
pdmartinez@informatica:~/Documentos/PyBDSF$ pip install setuptools
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: setuptools in /usr/lib/python3/dist-packages (45.2.0)
```

7. *python-casacore*.

```
pdmartinez@informatica:~/Documentos/PyBDSF$ pip install python-casacore
Defaulting to user installation because normal site-packages is not writeable
Collecting python-casacore
  Downloading python_casacore-3.5.2-cp38-cp38-manylinux_2_17_x86_64.manylinux201
4_x86_64.whl (35.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 35.1/35.1 MB 13.4 MB/s eta 0:00:00
```

8. *libboost-python*.

```
pdmartinez@informatica:~/Documentos/PyBDSF$ sudo apt-get install -y libboost-python-dev
[sudo] contraseña para pdmartinez:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
```

9. *libboost-numpy*.

```
pdmartinez@informatica:~/Documentos/PyBDSF$ sudo apt-get install -y libboost-numpy-dev
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
```

2.2. Instalación

Comenzamos con la instalación de la librería *bdsf*. Entre corchetes se escribe *ishell* para la instalación de la terminal interactiva.

```
pdmartinez@informatica:~/Documentos/PyBDSF$ pip install bdsf[ishell]
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: bdsf[ishell] in /home/pdmartinez/.local/lib/python3.8/site-packages (
1.10.3)
Requirement already satisfied: backports.shutil-get-terminal-size in /home/pdmartinez/.local/lib/pyt
```

Figura 2: Instalación de la terminal interactiva para la librería *bdsf*

Continuamos con la descarga de los archivos del repositorio de GitHub PyBDSF del usuario *lofar-astron*. Esto lo realizaremos mediante un *git clone* en la carpeta donde deseemos que se descargue el repositorio:

```
pdmartinez@informatica:~/Documentos$ git clone https://github.com/lofar-astron/PyBDSF
Clonando en 'PyBDSF'...
remote: Enumerating objects: 4358, done.
remote: Counting objects: 100% (598/598), done.
remote: Compressing objects: 100% (162/162), done.
remote: Total 4358 (delta 503), reused 452 (delta 434), pack-reused 3760
Recibiendo objetos: 100% (4358/4358), 8.40 MiB | 8.09 MiB/s, listo.
Resolviendo deltas: 100% (2706/2706), listo.
```

Figura 3: Descarga del repositorio

Entramos la carpeta que se creó con el *git clone* y realizamos la instalación de PyBDSF mediante el siguiente comando:

```
pdmartinez@informatica:~/Documentos/PyBDSF$ pip install .[ishell]
Defaulting to user installation because normal site-packages is not writeable
Processing /home/pdmartinez/Documentos/PyBDSF
  Installing build dependencies ... done
```

Figura 4: Instalación de PyBDSF con la terminal interactiva.

Al ejecutar este comando podría aparecer el siguiente error:

```
  note: This error originates from a subprocess, and is likely not a problem with pip.
  ERROR: Failed building wheel for bdsf
Failed to build bdsf
ERROR: Could not build wheels for bdsf, which is required to install pyproject.toml-based projects
WARNING: There was an error checking the latest version of pip.
```

Figura 5: Error de *build wheels*

Para corregirlo debemos ejecutar la siguiente instrucción:

```
pdmartinez@informatica:~/Documentos/PyBDSF_Tutorial$ pip wheel bdsf
Collecting bdsf
  Downloading bdsf-1.10.3-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (4.0 kB)
Collecting backports.shutil-get-terminal-size (from bdsf)
```

Figura 6: Construcción de Wheels para BDSF

Y procedemos a ejecutar la instrucción de la figura 4. Una vez realizado esto ya podemos ejecutar la instrucción *pybdsf* y entrar a la terminal interactiva.

```
pdmartinez@informatica:~/Documentos/PyBDSF$ pybdsf

PyBDSF version 1.10.3
=====================================================================
PyBDSF commands
  inp task ............ : Set current task and list parameters
  par = val ........... : Set a parameter (par = '' sets it to default)
                          Autocomplete (with TAB) works for par and val
  go .................. : Run the current task
  default ............. : Set current task parameters to default values
  tput ................ : Save parameter values
  tget ................ : Load parameter values
PyBDSF tasks
  process_image ....... : Process an image: find sources, etc.
  show_fit ............ : Show the results of a fit
  write_catalog ....... : Write out list of sources to a file
  export_image ........ : Write residual/model/rms/mean image to a file
PyBDSF help
  help command/task ... : Get help on a command or task
                          (e.g., help process_image)
  help 'par' .......... : Get help on a parameter (e.g., help 'rms_box')
  help changelog ...... : See list of recent changes
---------------------------------------------------------------------
```

Figura 7: Terminal interactiva de *PyBDSF*

3. Uso de PyBDSF

Podemos hacer uso de PyBDSF mediante dos formas: la terminal interactiva y mediante scripts de Python. Para probar la correcta ejecución de PyBDSF dejo algunos archivos .fits de prueba en el repositorio PyBDSF_Tutorial en la carpeta *fits*.

Para el análisis de estos archivos son necesarias algunas condiciones. En primer lugar el archivo no debe tener más extensiones además del mapa a analizar, y esta única extensión debe tener un header con ciertas características. El header que uso es el siguiente:

```
SIMPLE  =                    T / conforms to FITS standard
BITPIX  =                  -64 / array data type
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                  101
NAXIS2  =                  101
CTYPE1  = 'RA---TAN'           / WCS Projection Type 1
CUNIT1  = 'deg     '           / WCS Axis Unit 1
CRVAL1  =            109.3789 / WCS Ref Pixel Value 1
CDELT1  =       -0.0002777778 / WCS Pixel Scale 1
CRPIX1  =               379.5 / WCS Ref Pixel 1
CTYPE2  = 'DEC--TAN'           / WCS Projection Type 2
CUNIT2  = 'deg     '           / WCS Axis Unit 2
CRVAL2  =           37.75826 / WCS Ref Pixel Value 2
CDELT2  =        0.0002777778 / WCS Pixel Scale 2
CRPIX2  =               387.5 / WCS Ref Pixel 2
CTYPE3  = 'FREQ    '           / WCS Projection Type 3
CUNIT3  = 'Hz      '           / WCS Axis Unit 3
CRVAL3  =                 1.0 / WCS Ref Pixel Value 3
CDELT3  =                 1.0 / WCS Pixel Scale 3
CRPIX3  =                 1.0 / WCS Ref Pixel 3
CTYPE4  = 'STOKES  '           / WCS Projection Type 4
CUNIT4  = '        '           / WCS Axis Unit 4
CRVAL4  =                 1.0 / WCS Ref Pixel Value 4
CDELT4  =                 1.0 / WCS Pixel Scale 4
CRPIX4  =                 1.0 / WCS Ref Pixel 4
UNIT    = 'mJy/beam'           / Unit of map
END
```

Figura 8: Header del archivo .fits

3.1. Terminal interactiva

Realizando un ejemplo con el archivo "Const.fits" exploraremos cada una de las tareas que pueden realizarse en la terminal interactiva.

3.1.1. process_image()

El proceso *process_image* sirve para leer un archivo fits y detectar las fuentes en ella. Para acceder al menú del proceso (y al de los demás) escribimos un *inp* antes de la tarea, del siguiente modo:

```
BDSF [2]: inp process_image
--------> inp(process_image)
PROCESS_IMAGE: Find and measure sources in an image.
================================================================================================================================
filename ................. '': Input image file name
adaptive_rms_box ..... False : Use adaptive rms_box when determining rms and mean maps
advanced_opts ........ False : Show advanced options
atrous_do ............ False : Decompose Gaussian residual image into multiple scales
beam ................. None : FWHM of restoring beam. Specify as (maj, min, pos ang E of N) in degrees. E.g., beam = (0.06,
                               0.02, 13.3). None => get from header
flagging_opts ........ False : Show options for Gaussian flagging
frequency ............ None : Frequency in Hz of input image. E.g., frequency = 74e6. None => get from header.
interactive .......... False : Use interactive mode
mean_map .......... 'default': Background mean map: 'default' => calc whether to use or not, 'zero' => 0, 'const' => clipped
                               mean, 'map' => use 2-D map
multichan_opts ....... False : Show options for multi-channel images
output_opts .......... False : Show output options
polarisation_do ...... False : Find polarisation properties
psf_vary_do .......... False : Calculate PSF variation across image
rms_box .............. None : Box size, step size for rms/mean map calculation. Specify as (box, step) in pixels. E.g.,
                               rms_box = (40, 10) => box of 40x40 pixels, step of 10 pixels. None => calculate inside program
rms_map .............. None : Background rms map: True => use 2-D rms map; False => use constant rms; None => calculate inside
                               program
shapelet_do .......... False : Decompose islands into shapelets
spectralindex_do ..... False : Calculate spectral indices (for multi-channel image)
thresh ............... None : Type of thresholding: None => calculate inside program, 'fdr' => use false detection rate
                               algorithm, 'hard' => use sigma clipping
thresh_isl ............ 3.0 : Threshold for the island boundary in number of sigma above the mean. Determines extent of island
                               used for fitting
thresh_pix ............ 5.0 : Source detection threshold: threshold for the island peak in number of sigma above the mean. If
                               false detection rate thresholding is used, this value is ignored and thresh_pix is calculated
                               inside the program
```

Figura 9: Menú del proceso *process_image*

Donde iremos configurando cada parámetro uno por uno y ejecutamos la tarea con *go()*:

```
BDSF [3]: filename = 'Const.fits'

BDSF [4]: beam = (0.0016,0.0016,0)

BDSF [5]: adaptive_rms_box = 'True'

BDSF [6]: frequency = 2.72e11

BDSF [7]: go()
--> Opened 'Const.fits'
```

Figura 10: Parámetros a utilizar en *process_image*

También podemos ejecutar el proceso en una sola línea metiendo en los paréntesis cada uno de los parámetros separados por coma:

```
BDSF [2]: process_image(filename='Const.fits',beam=(0.0016,0.0016,0),
     ...: adaptive_rms_box = 'True',frequency = 2.27e11)
```

Figura 11: Ejecución de *process_image* en una sola línea.

Una vez ejecutado el proceso nos mostrará un resumen del resultado del análisis, la cantidad de islas y fuentes encontradas, el flujo total y otros datos de interés.

```
BDSF [7]: go()
--> Opened 'Const.fits'
Image size ................................ : (101, 101) pixels
Number of channels ........................ : 1
Number of Stokes parameters ............... : 1
Beam shape (major, minor, pos angle) .... : (1.60000e-03, 1.60000e-03, 0.0) degrees
Frequency of image ........................ : 272000.000 MHz
Number of blank pixels .................... : 0 (0.0%)
Flux from sum of (non-blank) pixels ..... : 2.188 Jy
--> Calculating background rms and mean images
--> Using adaptive scaling of rms_box
Derived rms_box (box size, step size) ... : (22, 7) pixels (small scale)
Derived rms_box (box size, step size) ... : (45, 10) pixels (large scale)
--> Size of rms_box larger than 1/4 of image size
--> Using constant background rms and mean
Value of background rms ................... : 2.86e-02 Jy/beam
Value of background mean .................. : 0.00027 Jy/beam
--> Expected 5-sigma-clipped false detection rate > fdr_ratio
--> Using FDR (False Detection Rate) thresholding
FDR threshold (replaces thresh_pix) ..... : 2.2484
Minimum number of pixels per island ..... : 12
Number of islands found ................... : 2
Fitting islands with Gaussians ........... : [==] 2/2
Total number of Gaussians fit to image .. : 2
Total flux density in model ............... : 2.256 Jy
--> Grouping Gaussians into sources
Number of sources formed from Gaussians   : 2
```

Figura 12: Resumen de los resultados al ejecutar *process_image*

3.1.2. show_fit()

El proceso *show_fit* mostrará de manera visual el resultado obtenido del análisis.

```
BDSF [3]: show_fit()
```

Figura 13: Comando para mostrar el resultado del análisis.

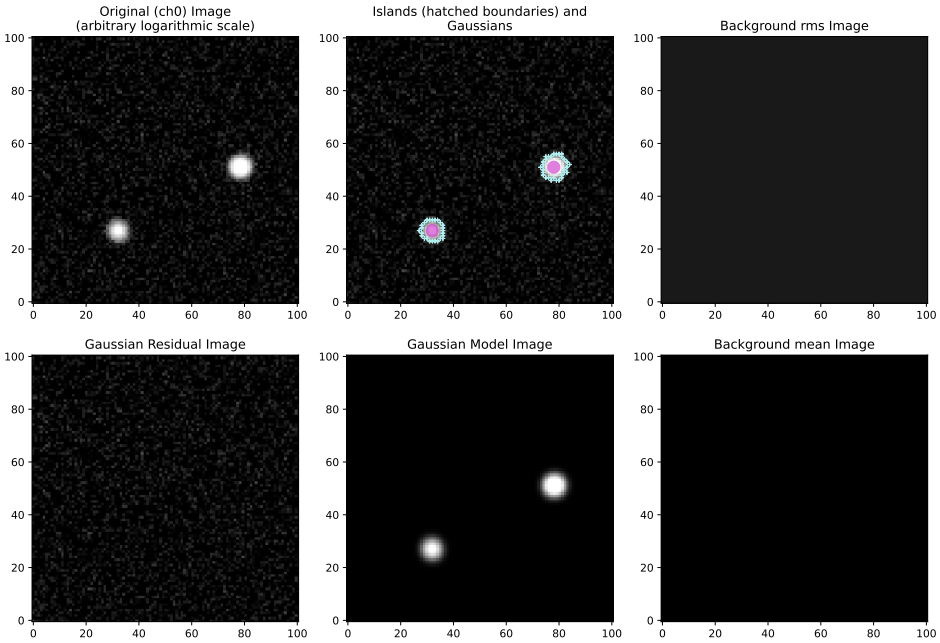

Figura 14: Ploteo del resultado del análisis.

El resultado de ejecutar *show_fit()* se muestra en la figura 14 y muestra seis mapas de los cuales mencionaremos:

1. **Original (ch0) image:** Imagen original, de la cual se hace el análisis.
2. **Islands (hatched boundaries) and Gaussians:** Es el mapa original donde se marcan las posiciones donde el ajuste encontró fuentes.
3. **Gaussian Model Image:** Son las fuentes aisladas sin el ruido de fondo.
4. **Gaussian Residual Image:** Es el ruido de fondo aislado.

3.1.3. export_image()

Este proceso sirve para exportar cualquiera de los mapas en formato “fits” o fomato “casa”, aunque usaremos más el primero. Del mismo modo, accedemos al menú mediante *inp*:

```
BDSF [4]: inp export_image
--------> inp(export_image)
EXPORT_IMAGE: Write an image to disk.
================================================================
outfile ............... None : Output file name. None => file is named
                               automatically; 'SAMP' => send to SAMP hub
                               (e.g., to TOPCAT, ds9, or Aladin)
clobber ............... False : Overwrite existing file?
img_format ........... 'fits': Format of output image: 'fits' or 'casa'
img_type ....... 'gaus_resid': Type of image to export: 'gaus_resid',
                               'shap_resid', 'rms', 'mean', 'gaus_model',
                               'shap_model', 'ch0', 'pi', 'psf_major',
                               'psf_minor', 'psf_pa', 'psf_ratio',
                               'psf_ratio_aper', 'island_mask'
mask_dilation ............ 0 : Number of iterations to use for island-mask
                               dilation. 0 => no dilation
pad_image ............ False : Pad image (with zeros) to original size
```

Figura 15: Menú del proceso *export_image*

Configuramos los parámetros correspondientes:

```
BDSF [8]: outfile = 'GausModel.fits'

BDSF [9]: img_type = 'gaus_model'

BDSF [10]: img_format = 'fits'

BDSF [11]: go()
--> Wrote file 'GausModel.fits'
```

Figura 16: Parámetros a utilizar en *export_image*

En este caso se generó el archivo *GausModel.fits* donde guarda la información del mapa *gaus_model* (es decir, las fuentes aisladas sin ruido) en formato *.fits*. Podemos visualizar el archivo resultante:

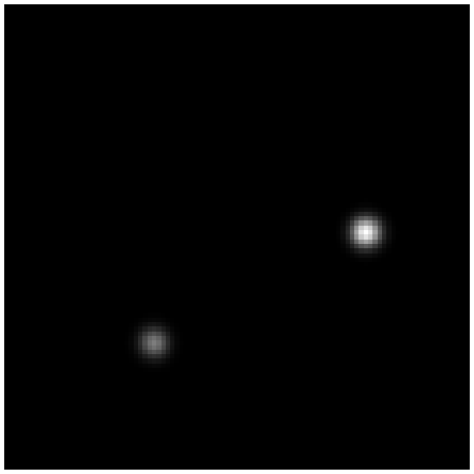

Figura 17: Contenido del archivo *GausModel.fits*

También podemos realizar este proceso en una sola línea de comandos. En este caso, obtenemos el mapa *gaus_resid*:

```
BDSF [12]: export_image(outfile = 'GausResid.fits',
img_type='gaus_resid',img_format =
      ...: 'fits')
--> Wrote file 'GausResid.fits'
```

Figura 18: Ejecución de *Export_image* en una sola línea.

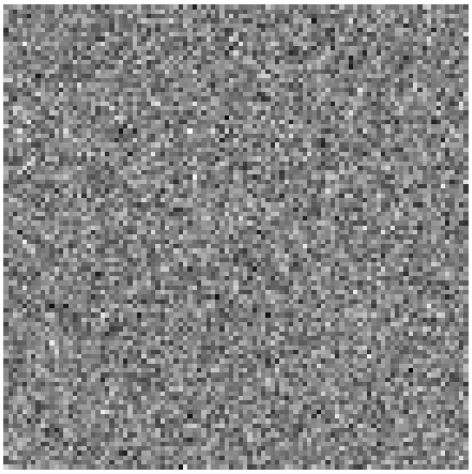

Figura 19: Contenido del archivo *GausResid.fits*

3.1.4. write_catalog()

Este proceso sirve para escribir un catálogo en distintos formatos donde se resuma la información obtenida. Al igual que siempre, podemos ingresar los parámetros uno a uno ejecutando primero *inp write_catalog* o en una sola línea de comandos:

```
---------> inp(write_catalog)
WRITE_CATALOG: Write the Gaussian, source, or shapelet list to a file.
================================================
outfile .... 'GausModel.fits': Output file name. None => file is named
                               automatically; 'SAMP' => send to SAMP hub
                               (e.g., to TOPCAT, ds9, or Aladin)
bbs_patches ........... None : For BBS format, type of patch to use: None
                               => no patches. 'single' => all Gaussians in
                               one patch. 'gaussian' => each Gaussian gets
                               its own patch. 'source' => all Gaussians
                               belonging to a single source are grouped
                               into one patch. 'mask' => use mask file
                               specified by bbs_patches_mask
bbs_patches_mask ...... None : Name of the mask file (of same size as input
                               image) that defines the patches if
                               bbs_patches = 'mask'
catalog_type ......... 'srl': Type of catalog to write:  'gaul' - Gaussian
                               list, 'srl' - source list (formed by
                               grouping Gaussians), 'shap' - shapelet list
                               (FITS format only)
clobber .............. False : Overwrite existing file?
correct_proj ......... True : Correct source parameters for image
                               projection (BBS format only)?
format ............... 'fits': Format of output catalog: 'bbs', 'ds9',
                               'fits', 'star', 'kvis', 'ascii', 'csv',
                               'casabox', or 'sagecal'
incl_chan ............ False : Include flux densities from each channel (if
                               any)?
incl_empty ........... False : Include islands without any valid Gaussians
                               (source list only)?
srcroot .............. None : Root name for entries in the output catalog
                               (BBS format only). None => use image file
                               name
```

Figura 20: Menú del proceso *write_catalog*

```
BDSF [14]: write_catalog(outfile = 'catalogo.csv',format='csv')
--> Wrote ASCII file 'catalogo.csv'
```

Figura 21: Ejecución en una línea de *write_catalog*

Donde obtenemos el archivo *catalog.csv* el cual contiene 47 columnas que se detallan aquí.

```
1  # Source list for Const.fits
2  # Generated by PyBDSM version 1.10.4.dev13+g9246686
3  # Reference frequency of the detection ("ch0") image: 2.27000e+11 Hz
4  # Equinox : 2000.0
5
6  # Source_id, Isl_id, RA, E_RA, DEC, E_DEC, Total_flux, E_Total_flux, Peak_flux, E_Peak_flux, RA_max, E_RA_max, DEC_max
7    0,    0, 109.50048346193363, 0.00001530313501, 37.65832820675424, 0.00001638351286, 0.75093879351593, 0.03820284060
8    1,    1, 109.48434358583819, 0.00000763930174, 37.66501817935706, 0.00000765973877, 1.50541200739932, 0.03755558910
```

Figura 22: Catalogo resultante.

3.2. Script de Python

Una de las principales bondades de PyBDSF es tener la opción de ser ejecutada mediante scripts de Python y ahorrarnos la ejecución paso a paso que implica el uso de la terminal. En el repositorio PyBDSF_Tutorial se encuentra el script *PyBdsf.py* que solamente requiere el nombre del archivo fits a analizar.

```
1  # PyBdsf.py
2  # Realizado por Luis Fernando Icú
3  # Email: luisfer200010@gmail.com
4  # GitHub: https://github.com/LuisIcu/PyBDSF_Tutorial
```

Figura 23: Header del script de Python

Al ejecutar el archivo será necesario ingresar el nombre del .fits (sin la extensión) y el nombre de salida. Se creará una carpeta con el nombre de salida donde guarda los siguientes archivos:

- Archivo fits de multiples extensiones donde se guardan los mapas obtenidos:
  1. Mapa original.
  2. Mapa booleano: los pixeles pueden tomar valores 1 o 0, 1 para cuando ahí se encuentra una fuente y 0 cuando no la haya.
  3. Modelo gaussiano: Las fuentes gaussianas aisladas.
  4. Residuo gaussiano: el ruido de fondo, al omitir las fuentes.
  5. Mapa de RMS: Mapa cuantificado del ruido de confusión del fondo mediante la raiz cuadrática media (RMS por sus siglas en inglés).
- Archivo .csv de 47 columnas con el nombre *prov.csv*.
- Archivo .csv con algunas columnas de interés:
  1. Isl_id: Número de isla.
  2. RA: Ascención recta.

3. DEC: Declinación.
4. Peak_Flux: Flujo en el pico.
5. E_Peak_Flux: Error del flujo en el pico.
6. Maj: Eje mayor del beam.
7. Min: Eje menor del beam.
8. S_Code: Código que indica si la fuente es única en su isla.

NOTA: para ejecutar el script el archivo .fits debe estar en la misma carpeta que el script. Si ya existe una carpeta con el nombre de salida se genera un error y no ejecuta nada.

Podemos ver un ejemplo de la ejecución:

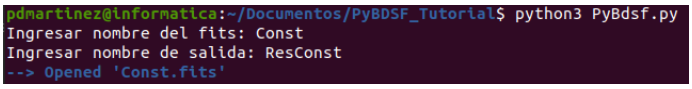

Figura 24: Ejecución del script y entrada de parámetros

Se crea una carpeta con el nombre de salida y ahí se guardan los archivos creados:

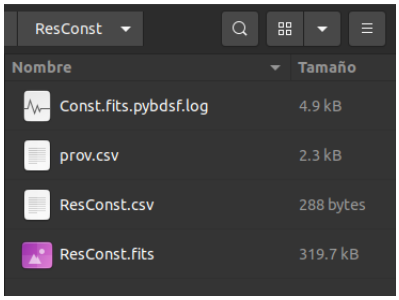

Figura 25: Carpeta creada.

Podemos ver el resultado del fits mediante ds9 y los valores obtenidos en el archivo csv.

Figura 26: Archivo csv resultante

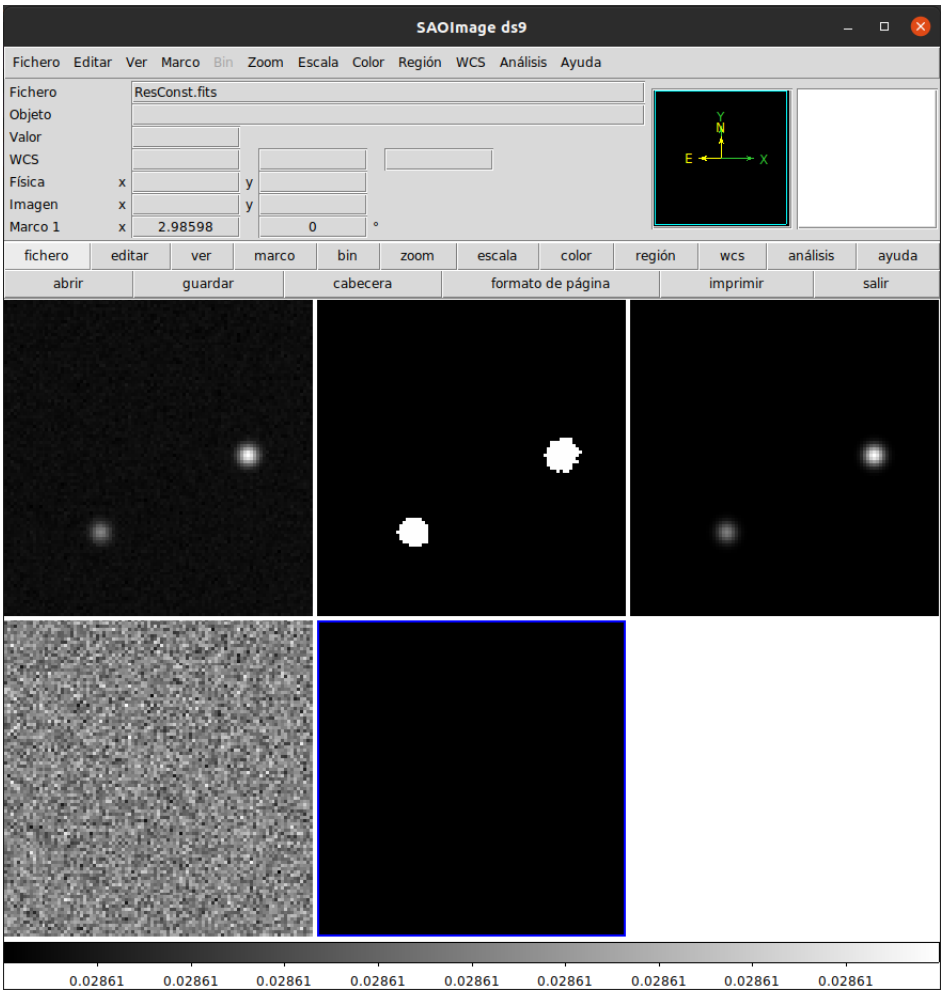

Figura 27: Las 5 extensiones creadas en un solo archivo fits