Manual para el programa Driver for Galfit on Galaxy Clusters (DGCG) que ajusta modelos de brillo superficial en imágenes para galaxias que se encuentran en cúmulos.

Los cúmulos de galaxias albergan alrededor de 100 – 1000 miembros de galaxias. Estudiar estas galaxias en estos lugares del universo son de gran interés ya que los cúmulos son conocidos por funcionar como laboratorios de galaxias. Se puede observar cambios evolutivos en las galaxias desde que entran al cúmulo hasta que se encuentran en él. Estos cambios son debido a la gran variedad de fenómenos físicos que ahí ocurren.

Figura 1. Cúmulo de galaxias. MCS J0416.1-2403

Un fenómeno conocido dentro de los cúmulos de galaxias es la relación densidad morfología. Esta relación indica que las galaxias tipo elípticas y S0s estarán en regiones donde la densidad de galaxias es mayor. Por el lado contrario, las galaxias espirales se encontrarán en las regiones donde la densidad de galaxias es menor.

Si uno desea analizar cada una de estas galaxias dependerá de las componentes de cada galaxia. Estas dependen de la morfología de cada tipo. En la clasificación del diagrama de diapasón de Hubble (ver siguiente figura), podemos encontrar 3 tipos básicos: galaxias espirales, S0s (o lenticulares) y Elípticas. Dentro de las galaxias espirales pueden tener una barra aunque solo sea un fenómeno transitorio de la misma.

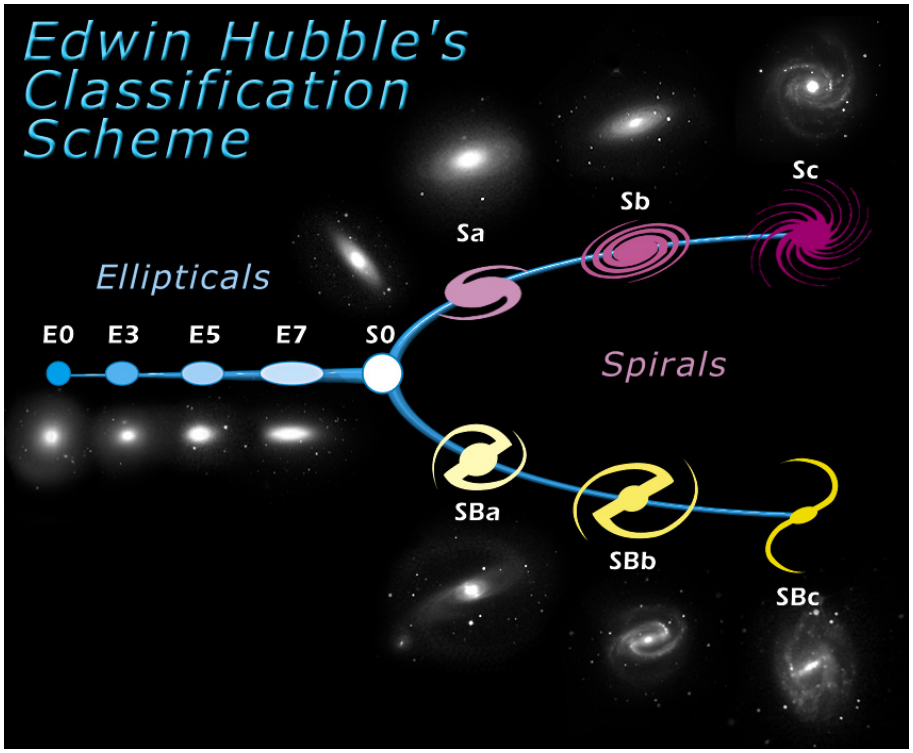


Figura 2. Diagrama de Diapasón de Hubble

Básicamente, elípticas tienen una componente, mientras que espirales y S0s tienen una componente de bulbo y disco. Las lenticulares se diferencian de las espirales en que estas últimas tienen brazos espirales. Estos dos tipos de galaxias pueden tener una componente extra el cual es una barra.

La cantidad de modelos para las galaxias dependerá de las componentes de cada una de estas. En el caso de elípticas, bulbos de galaxias espirales y bulbos de galaxias S0s se utiliza comúnmente la función Sérsic:

$$I(R) = I_e \exp\left\{-b_n\left[\left(\frac{R}{R_e}\right)^{1/n} - 1\right]\right\}$$

donde *Ie* es la intensidad al radio efectivo *Re* que encierra la mitad de la luz total del modelo. La constante b_n esta definida en términos del parámetro *n* el cual describe la forma de la ecuación del modelo de brillo superficial.

Para el caso del disco de las espirales y S0s se utiliza la función exponencial:

$$I(R) = I_0 \exp[-(R/h)]$$

Donde I_0 es la intensidad en el centro del modelo y h es la longitud de escala. La ecuación anterior se puede derivar de la ecuación de Sérsic cuando n=1

Por último para modelar las barras se utiliza la función gaussiana, que también puede ser derivado de la ecuación de la función Sérsic cuando n=0.5

Para obtener las funciones de cada una de las galaxias se necesita un programa que ajuste los modelos a cada una de las galaxias que se encuentren en la imagen. Para esto, ya existe el código GALFIT (Peng et al. 2002). Este permite analizar no solamente los modelos Sérsic, sino también los modelos exponencial, gaussiana, Nuker, moffat, Ferrer, King, PSF. Además GALFIT permite ajustar cualquier cantidad de modelos a una misma galaxia o en una misma imagen si así se desea.

Sin embargo, los ajustes de GALFIT son manuales. Esto significa que el usuario necesita indicar al programa cual galaxia de la imagen es la que se pretende ajustar. Además indicarle que regiones de la imagen no se desean ajustar. Se debe contar con un modelo de PSF para corregir efectos atmosféricos y finalmente proveerle a GALFIT un archivo con los parámetros iniciales del modelo o modelos de brillo superficial.

Usar GALFIT de forma manual para varias imágenes de cúmulos de galaxias con cúmulos de galaxias con miembros de 100 a 1000 galaxias se volvería una tarea imposible. Para resolver este problema, se ha creado el programa DGCG (Driver for GALFIT on Galaxy Clusters), en el cual este se encarga de automatizar GALFIT para que ajuste todas las galaxias que se encuentren en una misma imagen.

Lo que hace DGCG es ejecutar GALFIT dándole a este todos los archivos necesarios para que ajuste modelos a cada una de las galaxias que se encuentran en la imagen. Para esto, a DGCG se le debe de indicar en su archivo de configuración una imagen de cúmulos de galaxias, un catálogo de los objetos que se encuentran en ella y una lista de todos los modelos PSF de la imagen. Actualmente, DGCG permite configurar a GALFIT para que ajuste modelos ya sean de Sérsic o Sérsic + exponencial.

Cuando el programa termina, DGCG da un catálogo con los modelos ajustados junto con variables que son extraídas de los modelos y que no son calculados por GALFIT como lo son: razón bulbo a disco, magnitud total Bumpiness, Tidal. Esto permite al usuario realizar un mejor análisis de los modelos.

FUNCIONAMIENTO DE DGCG.

Instalación de programas y librerías.

Como DGCG es un script que está escrito para python 3, se requiere que esté instalado este (de preferencia a través de la paquetería Anaconda) y las siguientes librerías de python:

os, astropy, numpy, sys, scipy, subprocess, timeit, stat

Las cuales pueden ser instaladas bajo los siguientes comandos en Linux Ubuntu:

> sudo apt install python-numpy

o

> pip install numpy

tomando como ejemplo la instalación de la librería numpy. Nota: el simbolo “>” solo indica que se ejecuta en la terminal, no indica que sea parte del comando.

Archivos necesarios para DGCG

Para que DGCG funcione se necesita:
1.- Imagen de cúmulos de galaxias
2.- Archivo de restricciones de parámetros
3.- Catálogo de galaxias y estrellas de la imagen.
4.- Archivo de regiones (formato DS9) indicando regiones saturadas.
5.- Archivo de configuración de DGCG
6.- Archivo con las variables de salida de ajuste.

En el punto **1**, la imagen es la de un cúmulo de galaxias capturada de algún telescopio. Completamente reducida, calibrada y con los parámetros de EXPTIME y GAIN en el header para que puedan ser utilizados por GALFIT (para ver más detalles sobre la imagen consultar el manual del programa).

2 indica que es un archivo de restricciones a los parámetros de los modelos Sérsic y Sérsic + exponencial. Es el mismo archivo de restricciones de parámetros que solicita GALFIT y es igual para todos los ajustes de cada uno de los modelos.

En **3** y **4** son archivos que son generados por **pysex** el cual es un modulo separado de DGCG y el cual se explica más adelante. En **3** es catálogo que indica las posiciones, magnitudes, elipticidad, Kron radius, etc. **4** es un archivo para el programa de visualización de imágenes DS9, este archivo contiene regiones que encierran zonas de estrellas saturadas o cualquier región no deseada.

En **5** se indica el archivo de configuración de parámetros de configuración de DGCG. Este archivo contiene información de como debe de ejecutarse DGCG en la imagen: selección de modelos, tamaños de las regiones elípticas de cada objeto, calibración de la imagen. En otras palabras, contiene todo lo que necesita DGCG para que funcione. Detalles de este archivo se explican más adelante.

En el punto **6**, se indica que variables se desean en la salida del programa. DGCG calcula más de 70 variables y es posible que el usuario no necesite toda la información ya que depende de que análisis científico se pretende hacer. Para facilitar esto, se le puede indicar a DGCG que variables requiere de salida. También esta el programa **selcols.py** el cual ayuda a seleccionar que variables obtener en el catálogo de salida sin la necesidad de ejecutar DGCG de nuevo cada vez que se requiera un análisis diferente.

Creación del catalogo de objetos con pysex

El programa pysex.py crea un catálogo para que sea usado en DGCG (explicado en el punto 3 anteriormente). Básicamente lo que hace pysex es ejecutar Sextractor dos veces (tres en el caso de pysexbcg) con diferente configuración (bertin 1996) para obtener la mayor cantidad de objetos detectados .

Básicamente, Sextractor calcula la fotometría de cada uno de los objetos en la imagen. Como resultado, Sextractor da un catálogo con la información de cada objeto en la imagen. Para que este funciona se necesita un archivo de configuración para que sextractor analice la imagen.

Pysex ejecuta Sextractor con dos archivos de configuración diferentes: 1) para detectar objetos grandes y extendidos y 2) para detectar objetos débiles y pequeños. La combinación de dos catálogos de salida es el resultado de pysex y es el archivo de entrada para DGCG.

Funcionamiento de DGCG

para ejecutar DGCG solamente se necesita el siguiente comando (en Linux):

> ./dgcg dgcg.param

El archivo de parámetros de entrada (en este caso dgcg.param) se muestra a continuación:

```
Img             A85.fits             # Input image to be fitted
SexCat          a85.ppp               # The input sextractor catalog
SigImg          none                  # sigma image to use
PsfDir          psfs                  # PSF location (directory)
MagZpt          21.630               # Photometric zeropoint  (same as GALFIT)
PlateScale      0.68                  # Plate scale
FitFunc         sersic                # Fit a single sersic or do B/D decomposition?  (BD) or (sersic)
GalClas         0.6                  # Range of good galaxies: 0.0 (galaxy) to 1.0 (star) same as Sextractor
ConvBox         60                    # size of convolution box (pixels)
FitBox          6                     # times the galaxy size (KronRadius * Scale + 1) for fitting region. (6 recommend)
MagDiff         5                    # If object is x mag fainter than main source, then mask it instead of fit it
KronScale       1.5                  # Scale Factor by which Kron Ellipses are enlarged for neighbouring contaminating sou
SkyScale        1.6                  # Scale Factor by which Kron Ellipses are enlarged to calculate sky
Offset          20                   # Additional offset to scale factor (pix)
SkyWidth       40                     # width of sky annuli around sky kron ellipse to compute sky
NSer            1.5                  # Sersic index parameter for initial input for galaxies.
MaxFit          8                    # Max number of allowed contributing source for main source (for simultaneous fitting)
MagMax         18                    #  Max. mag limit of galaxy magnitude
FlagSex         4                     # Objects with sextractor Flags greater than this number are not fitted
ConsFile        constraints           # Parameter constraint file
Region          0                     # (0) Whole image catalog or, (1) region of the image
Boundary        0  0  0  0            # Boundary of the object region if (Region1) (xmin,ymin,xmax,ymax)
Split           5                     # Split image into how many parts along one axis?
SatRegionScale  2                     # Scale Factor by which Kron ellipse box are enlarged if AutoSatRegion  1
Ds9SatReg       badbox.reg            # user input ds9 saturation box region file (works if AutoSatRegion 0 )
FileOut         a85bdfits              # preposition name for output catalog files
HeadFlag        1                     # Write Header in output file
ColPar          selcol.param          # Selects which columns outputs to write in output
```

Figura 3. Ejemplo de archivo de configuración para DGCG

La explicación de cada parámetro importante se explicará durante la descripción del funcionamiento del algoritmo:

Una vez iniciado, DGCG ordena el catálogo de objetos que se proporcionó con la variable *SexCat*. El ordenamiento es de mayor a menor brillo para así dar prioridad a los objetos más brillantes cuando se ajusten los objetos con GALFIT.

Después el programa crea una imagen donde indica la posición y tamaño objetos por medio de elipses. Cada píxel dentro de una misma elipse contiene el mismo valor y es el que corresponde al número de catalogo de SexCat. Las elipses son generadas de la información geométrica de cada objeto y del valor de KronScale en el archivo de parámetros de entrada de DGCG. Esta imagen servirá como máscara para remover objetos externos que no se quieran ajustar con la galaxia de interés. Además se crea una imagen de mascara adicional que servirá para calcular el cielo exclusivamente (ver Figura 3).

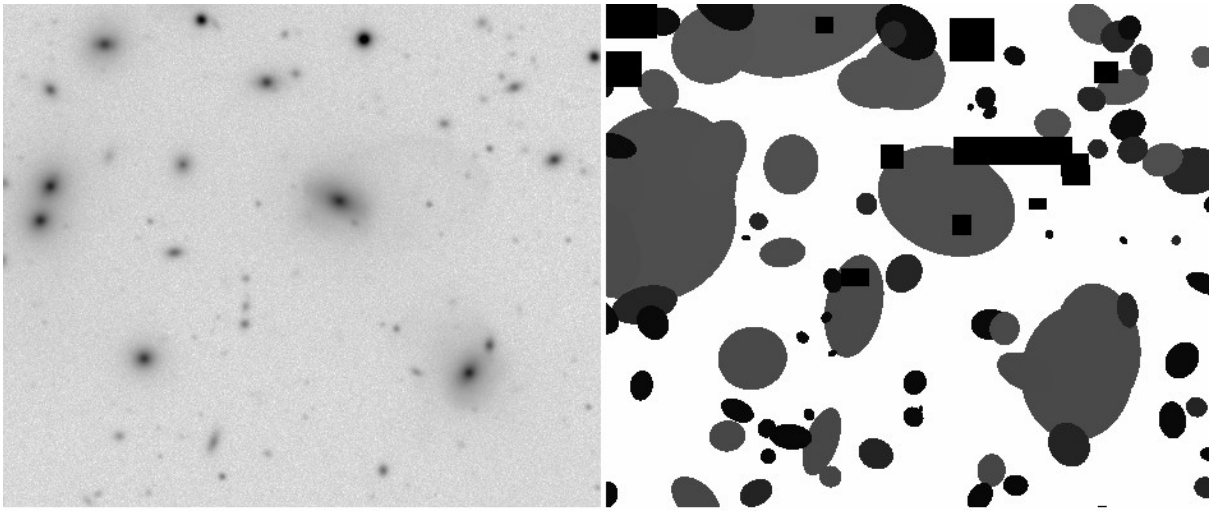


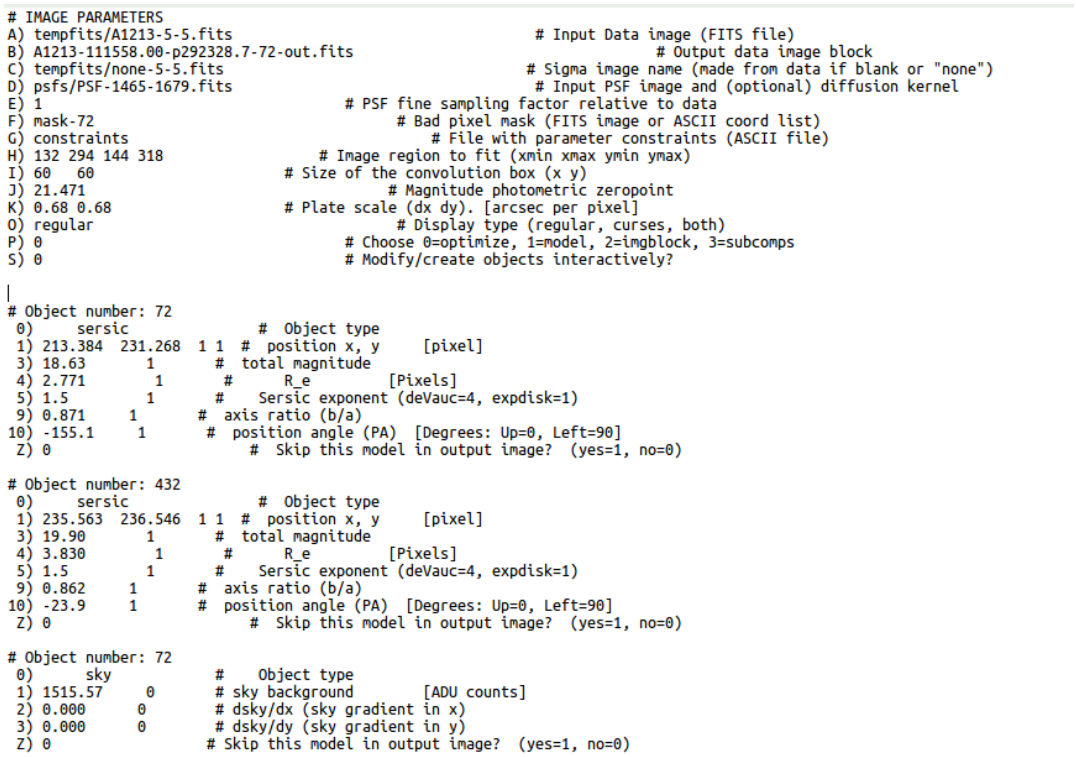
Figura 3. Izquierda: imagen de un cúmulo de galaxias. Derecha: imagen mascara del cúmulo de galaxias de la izquierda.

Una vez que se tienen las imágenes, estas son divididas acorde al parámetro Split, con la finalidad de acelerar el ajuste de GALFIT. La rápidez del programa aumenta considerablemente cuando la imagen contiene pocos píxeles.

Una vez que DGCG obtiene a partir del catálogo de entrada toda la información de cada objeto, este calcula el cielo de fondo para cada objeto con GALFIT, esto con el fin de obtener un mejor calculo de este y evitar degeneración de parámetros.

Una vez terminado el paso anterior, DGCG empieza a ajustar los modelos a las galaxias. El orden de ajuste empieza de la galaxia más brillante a la más débil. Este ordenamiento en el ajuste permite una mejor certidumbre en los parámetros. Para saber si una galaxia va a ser ajustada o no, esto dependerá de 3 filtros dados por las variables *MagMax*, *Class* y *Region* que indica el límite en magnitud de las galaxias que se van a ajustar, la clasificación estrella/galaxia y la región de la imagen en el cual se va a ajustar, respectivamente.

Si una galaxia fue seleccionada para ser ajustada, DGCG crea un archivo de configuración de GALFIT para cada objeto el cual le proporciona a este la información necesaria para que el programa ajuste modelos de brillo superficial ya sea para un ajuste Sersic o Sersic + Exponencial. Un ejemplo de archivo de configuración para GALFIT que crea DGCG se muestra a continuación:



```
# IMAGE PARAMETERS
A) tempfits/A1213-5-5.fits                          # Input Data image (FITS file)
B) A1213-111558.00-p292328.7-72-out.fits                        # Output data image block
C) tempfits/none-5-5.fits                          # Sigma image name (made from data if blank or "none")
D) psfs/PSF-1465-1679.fits                          # Input PSF image and (optional) diffusion kernel
E) 1                                # PSF fine sampling factor relative to data
F) mask-72                                # Bad pixel mask (FITS image or ASCII coord list)
G) constraints                                # File with parameter constraints (ASCII file)
H) 132 294 144 318                # Image region to fit (xmin xmax ymin ymax)
I) 60   60                        # Size of the convolution box (x y)
J) 21.471                                 # Magnitude photometric zeropoint
K) 0.68 0.68                      # Plate scale (dx dy). [arcsec per pixel]
O) regular                                # Display type (regular, curses, both)
P) 0                                # Choose 0=optimize, 1=model, 2=imgblock, 3=subcomps
S) 0                                # Modify/create objects interactively?

|
# Object number: 72
 0)     sersic              #  Object type
 1) 213.384  231.268  1 1  #  position x, y     [pixel]
 3) 18.63       1       #  total magnitude
 4) 2.771        1       #      R_e         [Pixels]
 5) 1.5          1       #    Sersic exponent (deVauc=4, expdisk=1)
 9) 0.871     1       #  axis ratio (b/a)
10) -155.1     1       #  position angle (PA)  [Degrees: Up=0, Left=90]
 Z) 0                      #  Skip this model in output image?  (yes=1, no=0)

# Object number: 432
 0)     sersic              #  Object type
 1) 235.563  236.546  1 1  #  position x, y     [pixel]
 3) 19.90       1       #  total magnitude
 4) 3.830        1       #      R_e         [Pixels]
 5) 1.5          1       #    Sersic exponent (deVauc=4, expdisk=1)
 9) 0.862     1       #  axis ratio (b/a)
10) -23.9     1       #  position angle (PA)  [Degrees: Up=0, Left=90]
 Z) 0                      #  Skip this model in output image?  (yes=1, no=0)

# Object number: 72
 0)      sky              #    Object type
 1) 1515.57     0       # sky background        [ADU counts]
 2) 0.000      0        # dsky/dx (sky gradient in x)
 3) 0.000      0        # dsky/dy (sky gradient in y)
 Z) 0                  # Skip this model in output image?  (yes=1, no=0)
```

Figura 4. Ejemplo de archivo de parámetros iniciales para GALFIT

Una vez que se tiene este archivo, DGCG procede a ejecutar GALFIT para que ajuste el modelaje de brillo superficial de las galaxias.

Si las galaxias están lo bastante cerca, el enmascaramiento no es suficiente para determinar la precisión de los parámetros ya que los flujos de las galaxias se mezclan. Para resolver esto, DGCG determina GALFIT si las galaxias vecinas a la galaxia de interés están lo suficientemente cerca para que están sean ajustadas simultáneamente o simplemente que sean enmascaradas. Si la galaxia vecina fue ajustada en un modelaje anterior, los parámetros de esta son usados como parámetros iniciales para el ajuste.

Una vez que GALFIT, terminó de ajustar todas las galaxias, este procede a calcular las variables de salida. Además de las ya calculadas por GALFIT, estos parámetros son: Magnitud Total, razón bulbo disco, parámetro de marea, bumpiness, señal a ruido entre otras.

Finalmente el programa DGCG termina, creando archivos de salida y provee un archivo que abre el programa DS9 con las imágenes de las galaxias, modelos y residuos para su posterior análisis por el usuario (ver figura 5).

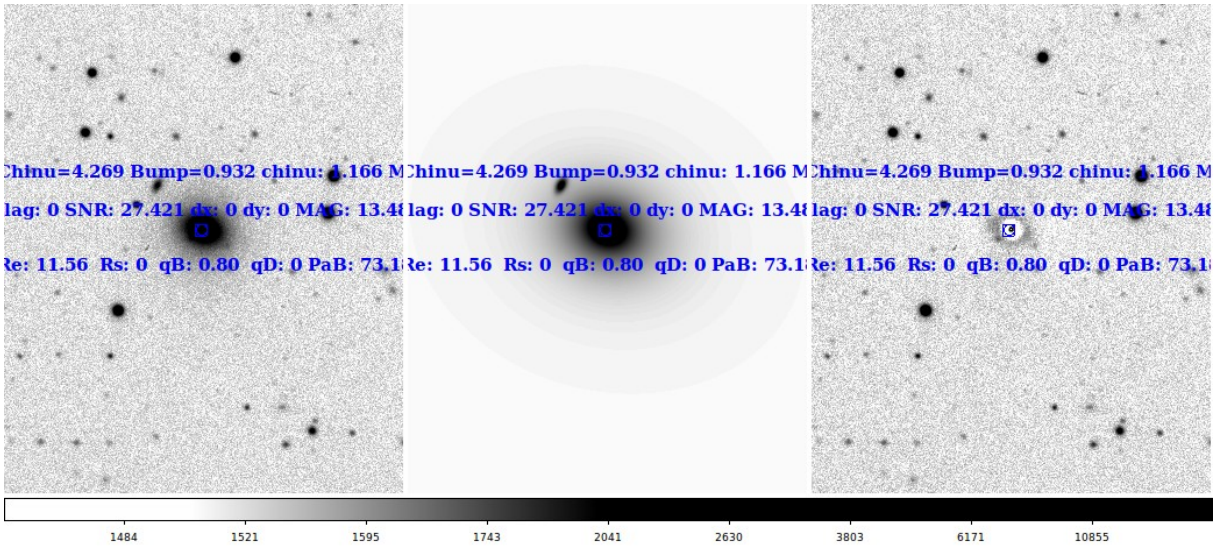


Figura 5. Ejemplo de imagen de salida de DGCG.

Archivos de Salida

Los nombres de archivos de salida están dados por la variable FileOut del archivo de entrada de DGCG. Los archivos de salida que genera DGCG en la salida son:

Principalmente el archivo FileOut.dgcg tiene variables de salida que son seleccionados por el usuario dependiendo de sus necesidades. El segundo archivo FileOut.dgcg.fits contiene todas las variables en formato table-fits (https://es.wikipedia.org/wiki/FITS). El archivo fits.log contiene información de todas variables de entrada y salida junto con la información del estado de cada ajuste durante su modelaje.

El programa guarda todos los modelos y archivos generados por GALFIT en la carpeta Outputs.

Ejemplo de ejecución

Un ejemplo de la ejecución con DGCG se muestra a continuación.
Desde la carpeta bin:

> dgcg dgcg.param > out

la parte de la linea de comandos “> out” no es necesaria, solamente indica que todas las variables de salida se guardan en el archivo out.

Un ejemplo de la salida de este programa es:

...
Creating table fits file..

Table fits file created

Erasing unnecesary files

DGCG had 87 success out of a total of 87

copying file obj-6 for refit. ReFit = True; FitFlag = 2
copying file obj-78 for refit. ReFit = True; FitFlag = 2
copying files to refit done..
Job took 15.76 minutes

Done everything!

Liga de DGCG a GitHub

El código de DGCG está abierto y se encuentra en la plataforma GitHub en la siguiente dirección:

https://github.com/canorve/DGCG

Referencias

1. Sandage, Allan. "The Classification of Galaxies: Early History and Ongoing Developments", Annual Review of Astronomy and Astrophysics, Vol. 43:581-624, 2005

2. Dressler, A. "The evolution of galaxies in clusters", Annual Review of Astronomy and Astrophysics, Vol. 22:185-222, 1984

3. Graham, A. & Driver, S. "A Concise Reference to (Projected) Sérsic $R^{1/n}$ Quantities, Including Concentration, Profile Slopes, Petrosian Indices, and Kron Magnitudes", Vol 22, 118-127, Publications of the Astronomical Society of Australia, 2005

4. Peng et al. "Detailed structural decomposition of galaxy images", The Astronomical Journal, Vol. 124: 266-293, 2002

5. Añorve, C. "Environments Effects on the physical properties of Galaxies: Clues to the formation of S0 galaxies" PhD Thesis, INAOE, 2012

Apéndice A:
Código de Software

```
#!/usr/bin/python3

import numpy as np
import sys
import os
import stat
import subprocess as sp
import os.path
from astropy.io import fits
import scipy
import scipy.special
from timeit import default_timer as timer

from lib import galfit
from lib import check
from lib import image
from lib import output
from lib import catfil


#################################################################################################
##
#                                DGCG                                    #
#                                                                        #
#                Driver for GALFIT on Cluster Galaxies                        #
#                                                                        #
#                DGCG is a wrapper script for GALFIT                         #
#                                                                        #
#                                                                        #
#                  written by Christopher Añorve                       #
#                                                                        #
# Note: DGCG was created from galfit.pl and galfit.cl (scripts from GALFIT package)       #
#################################################################################################
##

# Last Version: 5/Jul/2011
# MaxFit and MagDiff modified

# Last Version: 10/Mar/2011
# number of degrees of freedom is added to output

# Last Version: 1/Mar/2011
# flag = 4 is activated when a nan error is occurred

# Last Version: 24/Feb/2011
# a new pixel directory was added. Now the final parameters are computed within 1 kron radius

# Last Version: 19/Feb/2011
```

```
# JoinSexOut was modified. Now provide 99 mag to galaxies which were not fitted by GALFIT

# Last Version: 28/Jan/2011
# MakeOutput and Tidal functions were corrected. principally the computation of the SNR

# Last Version: 9/Dic/2010
# CheckSatRegion function was corrected. Also run time is print before to create final files

# Last Version: 4/Nov/2010
# Tidal, bumpiness, local chinu, snr are computed inside the Scale * Kron radius

# Last Version: 17/Oct/2016
# Program was translated to Python
# Now it is called pyDGCG

# Last Version: April/2017
# files and folders have a tree structure
# variables are stored in a Class


#################
#  main code:  #
#################

def main():

    global Version
    global StartRun,EndRun

    Version = "3.0  March/2018"


    if len(sys.argv[1:]) != 1:
        print ('Missing arguments')
        print ("Usage:\n %s [ConfigFile] " % (sys.argv[0]))
        print ("Example:\n %s Config.txt " % (sys.argv[0]))
        print ("DGCG Version: {} \n".format(Version))

        sys.exit()
    else:
        print ("DGCG Version: {} \n".format(Version))


    InFile= sys.argv[1]



# starting to count time
    StartRun = timer()


#   initialize default variables
```

```
################################################
#   creating a Object for input file parameters
    ParVar = galfit.ParamFile()
################################################


# read parameter file
    catfil.ReadFile(ParVar,InFile)

# verify parameters have sane values
    check.CheckSaneValues(ParVar)


########################################
### deleting any previous files run by DGCG

    runcmd = "rm {}".format(ParVar.Crashes)
    errrm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                stderr=sp.PIPE, universal_newlines=True)


    runcmd = "rm {}".format(ParVar.Fitted)
    errrm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                stderr=sp.PIPE, universal_newlines=True)


    runcmd = "rm inforegion"
    errrm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                stderr=sp.PIPE, universal_newlines=True)


    runcmd = "rm fits.log"
    errrm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                stderr=sp.PIPE, universal_newlines=True)


    runcmd = "rm fitlog.dgcg"
    errrm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                stderr=sp.PIPE, universal_newlines=True)

    runcmd = "rm objflags"
    errrm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                stderr=sp.PIPE, universal_newlines=True)


    runcmd = "rm psf.temp"
    errrm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                stderr=sp.PIPE, universal_newlines=True)


    runcmd = "rm {}".format(ParVar.OffsetPos)
    errrm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                stderr=sp.PIPE, universal_newlines=True)
```

```
    runcmd = "rm {}".format(ParVar.SexSort)
    errrm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                   stderr=sp.PIPE, universal_newlines=True)

    runcmd = "rm {}".format(ParVar.SexArSort)
    errrm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                   stderr=sp.PIPE, universal_newlines=True)

    runcmd = "rm {}".format(ParVar.SkyCrashes)
    errrm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                   stderr=sp.PIPE, universal_newlines=True)

#    runcmd = "rm {}".format(ParVar.SkyFitted)
#    errrm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
#                    stderr=sp.PIPE, universal_newlines=True)


    runcmd = "rm {}".format(ParVar.ListObjs)
    errrm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                   stderr=sp.PIPE, universal_newlines=True)


    runcmd = "rm {}*".format(ParVar.FileOut)
    errm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                    stderr=sp.PIPE, universal_newlines=True)

    runcmd = "rm galfit.*"
    errm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                    stderr=sp.PIPE, universal_newlines=True)

    runcmd = "rm {}*".format(ParVar.Ds9OutName)
    errm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                    stderr=sp.PIPE, universal_newlines=True)


#    runcmd = "rm -r {}".format(ParVar.InputDir)
#    errm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
#                     stderr=sp.PIPE, universal_newlines=True)

    runcmd = "rm -r {}".format(ParVar.SkyDir)
    errm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                    stderr=sp.PIPE, universal_newlines=True)

    runcmd = "rm -r {}".format(ParVar.OutputDir)
    errm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                    stderr=sp.PIPE, universal_newlines=True)

    runcmd = "rm -r {}".format(ParVar.TempDir)
    errm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                    stderr=sp.PIPE, universal_newlines=True)
```

```
    runcmd = "rm -r {}".format(ParVar.RunDir)
    errm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                  stderr=sp.PIPE, universal_newlines=True)

    runcmd = "rm -r {}".format(ParVar.ReRunDir)
    errm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                  stderr=sp.PIPE, universal_newlines=True)


    runcmd = "rm -r {}".format(ParVar.MaskDir)
    errm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                  stderr=sp.PIPE, universal_newlines=True)


    runcmd = "rm fit.log"
    errrm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                stderr=sp.PIPE, universal_newlines=True)



########################################


    (ParVar.NCol, ParVar.NRow) = image.GetAxis(ParVar.Img)
    (ParVar.ExpTime)           = image.GetExpTime(ParVar.Img)
    (ParVar.Gain)              = image.GetGain(ParVar.Img)
#    (ParVar.Rdnoise)           = image.GetRdnoise(ParVar.Img)


    ParVar.Total = catfil.CatArSort(ParVar)

    ParVar.Total = catfil.CatSort(ParVar)


##### segmentation mask

    if ((not(os.path.isfile(ParVar.SegFile))) or (ParVar.Overwrite == 1)):

        image.MakeImage(ParVar.SegFile, ParVar.NCol, ParVar.NRow)

#        if (ParVar.AutoSatRegion == 1):
#            catfil.MakeSatDs9(ParVar)

#       OldMakeMask(SegFile, SexArSort, KronScale,Ds9SatReg)
         image.MakeMask(ParVar.SegFile, ParVar.SexArSort, ParVar.KronScale,0,ParVar.Ds9SatReg)  # offset set
to 0
        image.MakeSatBox(ParVar.SegFile, ParVar.Ds9SatReg, ParVar.Total + 1, ParVar.NCol, ParVar.NRow)
    else:
        print("Using old mask image {} \n".format(ParVar.SegFile))


########### Sky segmentation annuli Mask  ##########
```

```
    if ((not(os.path.isfile(ParVar.SkyFile))) or (ParVar.Overwrite == 1)):

        image.MakeImage(ParVar.SkyFile, ParVar.NCol, ParVar.NRow)

#       if (ParVar.AutoSatRegion == 1):
#           catfil.MakeSatDs9(ParVar)

#       MakeSkyMask(SkyFile, SexArSort, SkyScale, Offset, Ds9SatReg) # Old

        image.MakeMask(ParVar.SkyFile, ParVar.SexArSort, ParVar.SkyScale, ParVar.Offset, ParVar.Ds9SatReg)
        image.MakeSatBox(ParVar.SkyFile, ParVar.Ds9SatReg, ParVar.Total + 1, ParVar.NCol, ParVar.NRow)


    else:
        print("Using old mask image {} \n".format(ParVar.SkyFile))


    catfil.UpdateSatFlags(ParVar)

################ THIS SECTION WAS REMOVED BECAUSE PixFile IS NOT NEEDED ANYMORE ##################
# This section is needed to compute final parameters
# check how this will be affected at the end

#    if ((not(os.path.isfile(PixFile))) or (Overwrite == 1)):

#       MakeImage(PixFile, NCol, NRow)

#       if (AutoSatRegion == 1):
#           MakeSatDs9(SexCat, SatRegionScale, NCol, NRow, Ds9SatReg)

#       OldMakeMask(PixFile, SexArSort, 1,Ds9SatReg)  # 1 Kron radius
#       MakeMask(PixFile, SexArSort, 1,0,Ds9SatReg)  # 1 Kron radius
#       MakeSatBox(PixFile, Ds9SatReg, Total + 1, NCol, NRow)
#    else:
#       print("Using old mask image {} \n".format(PixFile))

###################


### creating directories...

#    if not os.path.exists(ParVar.InputDir):
#       os.makedirs(ParVar.InputDir)

    if not os.path.exists(ParVar.MaskDir):
        os.makedirs(ParVar.MaskDir)

    if not os.path.exists(ParVar.OutputDir):
        os.makedirs(ParVar.OutputDir)

    if not os.path.exists(ParVar.SkyDir):
        os.makedirs(ParVar.SkyDir)
```

```
    if not os.path.exists(ParVar.RunDir):
        os.makedirs(ParVar.RunDir)


#    if not os.path.exists(ParVar.PixDir):
#        os.makedirs(ParVar.PixDir)

#    if not os.path.exists(ParVar.MaskPixDir):
#        os.makedirs(ParVar.MaskPixDir)


    XChunk = int(ParVar.NCol / ParVar.Split)
    YChunk = int(ParVar.NRow / ParVar.Split)


# REMOVED NOT NEEDED ANYMORE
# mask pixels

#    print ("Getting pixels from every object to remove masks \n")
#    GetPixels(SegFile,SexSort,KronScale,PixPrefix,XChunk,YChunk,Buffer)

#    errpix = system("mv PixPrefix* MaskPixDir/\. ")
#    CheckError(errpix)

# object pixels

#    print ("Getting pixels from every object to compute final parameters \n")
#    GetPixels(PixFile,SexSort,1,PixPrefix,XChunk,YChunk,Buffer)

#    errpix = system("mv PixPrefix* PixDir/\. ")
#    CheckError(errpix)
#########################################################

    print("Creating Ds9 Box region of all objects \n")
    catfil.BoxDs9(ParVar)


# splitting image files
    print("Splitting images \n")
    image.SplitImage(ParVar)

#    f = open("psf.temp", "w")
#    optarg = "{}/PSF*.fits".format(PsfDir)
#    errpsf = sp.call(["ls", optarg], stdout=f)

    runcmd = "ls {}/PSF*.fits > psf.temp".format(ParVar.PsfDir)
    errpsf = sp.run([runcmd], shell=True, stdout=sp.PIPE,
            stderr=sp.PIPE, universal_newlines=True)


#    f.close()
# CheckError(errpsf)
```

```
    print("====================== DGCG In a NutShell ========================== \n")
    print(" This script takes the pySEx output, and formats it for GALFIT.  \n")
    print(" DGCG: Driver for GALFIT on Cluster Galaxies                      \n")
    print(" Created by Christopher Añorve et al.                           \n")
    print("=================================================================== \n")


    flog = open(ParVar.LogFile, "w")

# print DGCG options in Log file

    output.PrintVar(ParVar,flog)

#    OffsetFile = "OffsetPos"

    fobjs = open(ParVar.ListObjs, "w")

    fout2 = open(ParVar.OffsetPos, "w")

    fout3 = open(ParVar.Crashes, "w")
    fout4 = open(ParVar.Fitted, "w")


#    fskycrash = open(dgcg.SkyCrashes, "w")
#    fskyfit = open(ParVar.SkyFitted, "w")


############################
# defining a object Class to store all the variables
   Obj = galfit.Object()
############################

###############  Read in sextractor sorted data ################

    Obj.Num, Obj.RA, Obj.Dec, Obj.XPos, Obj.YPos, Obj.Mag, Obj.Kron, Obj.FluxRad, Obj.IsoArea, Obj.AIm,
Obj.E, Obj.Theta, Obj.Background, Obj.Class, Obj.Flag, Obj.XMin, Obj.XMax, Obj.YMin, Obj.YMax,
Obj.XSMin, Obj.XSMax, Obj.YSMin, Obj.YSMax = np.genfromtxt(
            ParVar.SexSort, delimiter="", unpack=True)  # sorted

##
   Obj.Angle = Obj.Theta - 90
   Obj.AR = 1 - Obj.E
   Obj.RKron = ParVar.KronScale * Obj.AIm * Obj.Kron
   Obj.Sky = Obj.Background

   Obj.Num = Obj.Num.astype(int)
   Obj.Flag = Obj.Flag.astype(int)

# other stuff:
```

```
#    Tot = len(Obj.Num)
    Tot = ParVar.Total

    Obj.Sersic = [ParVar.NSer] * Tot
    Obj.Sersic = np.array(Obj.Sersic)

    Obj.RSky = ParVar.SkyScale * Obj.AIm * Obj.Kron + ParVar.Offset + ParVar.SkyWidth
    Obj.RKron = ParVar.KronScale * Obj.AIm * Obj.Kron


    masky = Obj.RSky <= 0
    if masky.any():
        Obj.RSky[masky] = 1

    maskron = Obj.RKron <= 0
    if maskron.any():
        Obj.RKron[maskron] = 1

    Obj.SkyFlag = [True] * Tot
    Obj.SkyFlag = np.array(Obj.SkyFlag)

    Obj.Neighbors = Obj.Num


# subpanel stuff:

    Obj.IX = (Obj.XPos / XChunk) + 1
    Obj.IY = (Obj.YPos / YChunk) + 1

    Obj.IX = Obj.IX.astype(int)
    Obj.IY = Obj.IY.astype(int)


    Obj.XMin = Obj.XMin.astype(int)
    Obj.XMax = Obj.XMax.astype(int)
    Obj.YMin = Obj.YMin.astype(int)
    Obj.YMax = Obj.YMax.astype(int)

    Obj.XSMin = Obj.XSMin.astype(int)
    Obj.XSMax = Obj.XSMax.astype(int)
    Obj.YSMin = Obj.YSMin.astype(int)
    Obj.YSMax = Obj.YSMax.astype(int)



    maskblkx = Obj.IX > ParVar.Split
    if maskblkx.any():
        Obj.IX[maskblkx]= Obj.IX[maskblkx] -1


    maskblky = Obj.IY > ParVar.Split
    if maskblky.any():
        Obj.IY[maskblky]= Obj.IY[maskblky] -1
```

```
#   Make sure the object coordinate in the subpanel is correct

# create arrays

    Obj.XBuffer=np.array([0]*Tot)
    Obj.YBuffer=np.array([0]*Tot)


    maskix = Obj.IX == 1
    if maskix.any():
        Obj.XBuffer[maskix] = 0

    maskix = Obj.IX != 1
    if maskix.any():
        Obj.XBuffer[maskix] = ParVar.Buffer

    maskiy = Obj.IY == 1
    if maskiy.any():
        Obj.YBuffer[maskiy] = 0

    maskiy = Obj.IY != 1
    if maskiy.any():
        Obj.YBuffer[maskiy] = ParVar.Buffer

# Obj.OFFX and Obj.OFFY transform coordinates
#  from big image to tile image --> IM-X-Y.fits
    Obj.OFFX = (Obj.IX - 1) * XChunk - Obj.XBuffer
    Obj.OFFY = (Obj.IY - 1) * YChunk - Obj.YBuffer


#####################################################################
#####################################################################
# creating empty arrays

#    Obj.gXMIN = np.array([0]*Tot)
#    Obj.gXMAX = np.array([0]*Tot)
#    Obj.gYMIN = np.array([0]*Tot)
#    Obj.gYMAX = np.array([0]*Tot)


    XSize = Obj.XMax - Obj.XMin
    YSize = Obj.YMax - Obj.YMin

#   enlarge fit area

    XSize = ParVar.FitBox * XSize
    YSize = ParVar.FitBox * YSize

##  30 pixels is the minimum area to fit (arbitrary number):

    masksize = XSize < 30
```

```
if masksize.any():
   XSize[masksize] = 30

masksize = YSize < 30

if masksize.any():
   YSize[masksize] = 30

#  Calculate the (x,y) position of the current object relative to
#  the tile in which it lives.

XFit = Obj.XPos - Obj.OFFX
YFit = Obj.YPos - Obj.OFFY

#  Calculate fitting box needed to plug into galfit header:

XLo = XFit - XSize / 2
XLo = XLo.astype(int)

maskxy = XLo <= 0
if maskxy.any():
   XLo[maskxy] = 1


XHi = XFit + XSize / 2
XHi = XHi.astype(int)

maskxy = XHi > ParVar.NCol  #This does not affect the code at all
if maskxy.any():
   XHi[maskxy] = ParVar.NCol


YLo = YFit - YSize / 2
YLo = YLo.astype(int)

maskxy = YLo <= 0
if maskxy.any():
   YLo[maskxy] = 1


YHi = YFit + YSize / 2
YHi = YHi.astype(int)

maskxy = YHi > ParVar.NRow  # same as above but for y axis
if maskxy.any():
   YHi[maskxy] = ParVar.NRow


# Calculate information needed to plug back into the image header
# at the end of the fit.

Obj.gXMIN = XLo + Obj.OFFX  # The [gXMIN:gXMAX,gYMIN:gYMAX] of the box
```

```
    Obj.gXMAX = XHi + Obj.OFFX  # relative to the big image from which
    Obj.gYMIN = YLo + Obj.OFFY  # the current tile was extracted.
    Obj.gYMAX = YHi + Obj.OFFY

    Obj.gOFFX = Obj.gXMIN - 1
    Obj.gOFFY = Obj.gYMIN - 1


################################################################
################################################################
# XLo, YLo, XHi, YHi correspond to the small image
# gXMIN, gYMIN, gXMAX, gYMAX correspond to the big image (original)
################################################################
################################################################


# Creating empty arrays for output variables
    Obj.ra = np.array(["none"]*Tot, dtype=object)
    Obj.dec = np.array(["none"]*Tot, dtype=object)

    Obj.InputImage = np.array(["none"]*Tot,dtype=object)
    Obj.OutIm = np.array(["none"]*Tot,dtype=object)
    Obj.Objx = np.array(["none"]*Tot,dtype=object)
    Obj.PPPnum = np.array([0]*Tot)
    Obj.Restart = np.array(["none"]*Tot,dtype=object)
    Obj.Cluster = np.array(["none"]*Tot,dtype=object)


    Obj.PosXSer = np.array([0.0]*Tot)
    Obj.ErPosXSer = np.array([0.0]*Tot)
    Obj.PosYSer = np.array([0.0]*Tot)
    Obj.ErPosYSer = np.array([0.0]*Tot)
    Obj.MagSer =  np.array([99.9]*Tot)
    Obj.ErMagSer = np.array([0.0]*Tot)
    Obj.ReSer = np.array([0.0]*Tot)
    Obj.ErReSer = np.array([0.0]*Tot)
    Obj.NSer = np.array([0.0]*Tot)
    Obj.ErNSer = np.array([0.0]*Tot)
    Obj.AxisSer = np.array([0.0]*Tot)
    Obj.ErAxisSer = np.array([0.0]*Tot)
    Obj.PaSer = np.array([0.0]*Tot)
    Obj.ErPaSer = np.array([0.0]*Tot)
    Obj.KSer = np.array([0.0]*Tot)

    Obj.MeanMeSer = np.array([99.9]*Tot)
    Obj.ErMeanMeSer = np.array([99.9]*Tot)
    Obj.MeSer = np.array([99.9]*Tot)
    Obj.ErMeSer = np.array([99.9]*Tot)


    Obj.PosXExp = np.array([0.0]*Tot)
    Obj.ErPosXExp = np.array([0.0]*Tot)
    Obj.PosYExp = np.array([0.0]*Tot)
```

```
    Obj.ErPosYExp = np.array([0.0]*Tot)

    Obj.MagExp = np.array([99.9]*Tot)
    Obj.ErMagExp = np.array([99.9]*Tot)

    Obj.RsExp = np.array([0.0]*Tot)
    Obj.ErRsExp = np.array([0.0]*Tot)
    Obj.AxisExp = np.array([0.0]*Tot)
    Obj.ErAxisExp = np.array([0.0]*Tot)
    Obj.PaExp = np.array([0.0]*Tot)
    Obj.ErPaExp = np.array([0.0]*Tot)
    Obj.MeanMeExp =np.array([99.9]*Tot)
    Obj.MeExp = np.array([99.9]*Tot)
    Obj.ErMeanMeExp =np.array([99.9]*Tot)
    Obj.ErMeExp = np.array([99.9]*Tot)
    Obj.MsExp = np.array([99.9]*Tot)
    Obj.ErMsExp = np.array([99.9]*Tot)

    Obj.Sky = np.array([0.0]*Tot)
    Obj.ErSky = np.array([0.0]*Tot)

    Obj.MagTotal = np.array([99.9]*Tot)
    Obj.ErMagTotal = np.array([99.9]*Tot)

    Obj.BulgeTotal = np.array([0.0]*Tot)
    Obj.ErBt = np.array([0.0]*Tot)
    Obj.ChiNu = np.array([0.0]*Tot)
    Obj.Tidal = np.array([0.0]*Tot)
    Obj.ObjChiNu = np.array([0.0]*Tot)
    Obj.Bump = np.array([0.0]*Tot)
#    Obj.MeanMesky = np.array([99.9]*Tot) # eliminado
    Obj.SNR = np.array([0.0]*Tot)
    Obj.NDof = np.array([0]*Tot)
    Obj.FitFlag = np.array([0]*Tot)
    Obj.ReFit =  np.array([False]*Tot)   #rerun if didn't fit
############################


##################
# posible to remove
#    print("Finding Neighbors for every object \n")

#    FindNeighbors()  # maybe remove this function
############

# 0: Sort catalog and create segmentation images
# 1: Execute everything compute sky, create input files, run galfit and create output file;
# 2: only computes sky
# 3: Execute everything except creation of output file


    if ParVar.Execute != 0:
```

```
        #   Sky fitting
        #   DGCG computes the sky first and leaves it fixed for galaxy fitting.

        print("DGCG is going to compute sky \n")


# remove fit.log before to compute sky
        runcmd = "rm fit.log"
        errrm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                    stderr=sp.PIPE, universal_newlines=True)

# remove fit.log before to compute sky
        runcmd = "rm galfit.*"
        errrm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                    stderr=sp.PIPE, universal_newlines=True)



###############
        galfit.RunSky(ParVar,Obj)
###############


        runcmd = "mv fit.log {}/fit.log.sky".format(ParVar.SkyDir)
        errmv = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                    stderr=sp.PIPE, universal_newlines=True)
#         CheckError(errmv)

        runcmd = "mv Sky-* {}/.".format(ParVar.SkyDir)
        errmv = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                    stderr=sp.PIPE, universal_newlines=True)


        runcmd = "mv galfit.* {}/.".format(ParVar.SkyDir)
        errmv = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                    stderr=sp.PIPE, universal_newlines=True)
#         CheckError(errmv)

        print("Done sky fitting \n")


        if (ParVar.Execute != 2):

#############################################################
            # here comes the serious stuff:
            galfit.DGCG(ParVar,Obj,flog,fobjs,fout2,fout3,fout4)
#############################################################


## removed after the introduction of RunDir
#           runcmd = "mv obj-* {}/.".format(ParVar.InputDir)
#           errmv = sp.run([runcmd], shell=True, stdout=sp.PIPE,
#                       stderr=sp.PIPE, universal_newlines=True)
```

```
#       CheckError(errmv)
#####################


          runcmd = "mv sigma-* {}/.".format(ParVar.MaskDir)
          errmv = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                         stderr=sp.PIPE, universal_newlines=True)
#       CheckError(errmv)

#           runcmd = "mv out-* {}/.".format(ParVar.OutputDir)
#           errmv = sp.run([runcmd], shell=True, stdout=sp.PIPE,
#                          stderr=sp.PIPE, universal_newlines=True)
#       CheckError(errmv)

          runcmd = "mv galfit.* {}/.".format(ParVar.OutputDir)
          errmv = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                         stderr=sp.PIPE, universal_newlines=True)
#       CheckError(errmv)

#           runcmd = "mv mask-* {}/.".format(ParVar.InputDir)
#           errmv = sp.run([runcmd], shell=True, stdout=sp.PIPE,
#                          stderr=sp.PIPE, universal_newlines=True)
#       CheckError(errmv)

          runcmd = "mv *-out.fits {}/.".format(ParVar.OutputDir)
          errmv = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                         stderr=sp.PIPE, universal_newlines=True)
#       CheckError(errmv)

#           mkdir = "{}/{}".format(ParVar.InputDir, ParVar.PsfDir)
#           if not os.path.exists(mkdir):
#               os.makedirs(mkdir)

#           runcmd = "cp {}/* {}/.".format(ParVar.PsfDir, mkdir)
#           errcp = sp.run([runcmd], shell=True, stdout=sp.PIPE,
#                          stderr=sp.PIPE, universal_newlines=True)
#       CheckError(errcp)

#           runcmd = "cp {} {}/.".format(ParVar.ConsFile, ParVar.InputDir)
#           errcp = sp.run([runcmd], shell=True, stdout=sp.PIPE,
#                          stderr=sp.PIPE, universal_newlines=True)
#       CheckError(errcp)

          print("Done GALFITting XD \n")



#####################################################################

# Closing files :

    flog.close()
    fout2.close()
```

```
fout3.close()
fout4.close()
fobjs.close()

#fskycrash.close()
#fskyfit.close()

outflag=False
finalflag=False
posflag=False  #change to True
joinflag=False

if (ParVar.Execute != 3 and ParVar.Execute != 2 and ParVar.Execute != 0):

   print("Creating outputs files \n")


   outflag = output.ReadFitlog2("fitlog.dgcg")

   if (outflag == True):

      if (ParVar.FitFunc == "BD"):
         ParVar.BtFile = ParVar.FileOut + ".bd"

      else:
         ParVar.BtFile = ParVar.FileOut + ".ser"

      finalflag = output.MakeOutput("fitlog.dgcg", ParVar, Obj)

      if (finalflag == True):

#           posflag = output.PosCor(OffsetFile, ParVar.BtFile)  # correct positions ## Deprecated

         posflag=True

         if (posflag == True):

#             ParVar.SexOut = ParVar.FileOut + ".dgcg"
            ParVar.SexOut = ParVar.BtFile + ".dgcg"

            # Join output file with sextractor catalog only for the fitted
            # objects


            joinflag = output.JoinSexOut(ParVar, Obj)


            FilColOut = ParVar.FileOut + ".dgcg"

            outcolflag=output.SelectColumns(ParVar.SexOut, ParVar.ColPar, FilColOut, ParVar.HeadFlag)


            if (joinflag == True):
```

```
                print("Ascii output file was succesfully created\n")

                ParVar.TableFits = ParVar.FileOut + ".dgcg.fits"

                print("Creating table fits file.. \n")

                outfitsflag = output.Ascii2Fits(ParVar.SexOut, ParVar.TableFits)

                print("Table fits file created\n")



            else:
                print("Can't create Ascii final output file \n")

        else:
            print("Can't create dgcg output file \n")

    else:
        print("Can't create bt output file \n")

else:
    print("Can't create output files \n")


# erasing files

  if (ParVar.Erase == 1):

    print("Erasing unnecesary files \n")

#     runcmd = "rm -r {}".format(ParVar.TempDir)
#     errm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
#                   stderr=sp.PIPE, universal_newlines=True)
    # CheckError(errm)

    os.remove("psf.temp")

    os.remove("inforegion")

    os.remove("ListObjs")

    os.remove("objflags")

    os.remove(ParVar.OffsetPos)

    os.remove(ParVar.SkyCrashes)

    os.remove(ParVar.Crashes)
    os.remove(ParVar.Fitted)
```

```
#   $errno = system("rm $InputDir/mask-*");
#   CheckError($errno);

        runcmd = "rm {}/Sky-*".format(ParVar.SkyDir)
        errm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                  stderr=sp.PIPE, universal_newlines=True)
        # CheckError(errm)

        runcmd = "rm {}/galfit.*".format(ParVar.SkyDir)
        errm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                  stderr=sp.PIPE, universal_newlines=True)
        # CheckError(errm)


        runcmd = "rm -r {}".format(ParVar.SkyDir)
        errm = sp.run([runcmd], shell=True, stdout=sp.PIPE,
                  stderr=sp.PIPE, universal_newlines=True)




    if (ParVar.Execute != 0 and ParVar.Execute != 2):

        GalTot = ParVar.Failures + ParVar.Success

        print("DGCG had {} success out of a total of {} \n".format(ParVar.Success, GalTot))



    if not os.path.exists(ParVar.ReRunDir):
        os.makedirs(ParVar.ReRunDir)


    maskrun= Obj.ReFit == True


    ind=np.where(maskrun == True)

    indx=ind[0]

    for idx in enumerate(indx):

        objid = Obj.Num[idx[1]]

        parfile =  "obj" + "-" + str(objid)

                pmsg="copying file {} for refit. ReFit = {}; FitFlag = {} ".format(parfile, Obj.ReFit[idx[1]],
Obj.FitFlag[idx[1]])
        print(pmsg)

        dirparfile =  ParVar.RunDir + "/" + parfile
```

```
        redirparfile =  ParVar.ReRunDir + "/" + parfile

        runcmd = "cp {} {}".format(dirparfile,redirparfile)
        errcp = sp.run([runcmd], shell=True, stdout=sp.PIPE,stderr=sp.PIPE, universal_newlines=True)

    print("copying files to refit done..")


#################################
#################################
#################################

# computing running time

    EndRun = timer()

    RunTime = EndRun - StartRun

    RunTimeMin = RunTime/60

    RunTimeHour = RunTimeMin/60

    RunTimeDays = RunTimeHour/24


    if (RunTimeDays >= 1 ):

        print ("Job took {0:.2f} days \n".format(RunTimeDays))


    elif (RunTimeHour >= 1):

        print ("Job took {0:.2f} hours \n".format(RunTimeHour))


    elif (RunTimeMin >= 1):

        print ("Job took {0:.2f} minutes \n".format(RunTimeMin))

    else:
        print ("Job took {0:.2f} seconds \n".format(RunTime))



    print ("Done everything! \n")



#end of program
if __name__ == '__main__':
    main()
```