Tutorial 2: HST/WFC3 Observations of Local Compact Galaxies

Name: Felix Martinez
Partner: Adrian Martinez
Date: 02/25/2021

1 Overview

This Tutorial will provide us ample knowledge in the process of transforming raw image data from the *Hubble Space Telescope* (*HST*) and reducing these images into observations suitable for scientific analysis. We will be reducing the data using calwf3 and astrodrizzle software that was developed by the Space Telescope Science Institute for initial processing, then begin using dithered exposures. The calwf3 pipeline is designed to be used with the Wide Field Camera 3 installed on *HST* while astrodrizzle is a package from the more general DrizzlePac software that helps reduce images from multiple *HST* instruments.

The object we will be focusing on is an early-type elliptical galaxy in the local Universe, UGC 2698. This galaxy was observed on Aug 26, 2013 with *HST* WFC3. We will be working with the IR/F160W images taken of this galaxy for reduction.

2 Results

Below we summarize our work for each section of the tutorial and address the questions posed in the manual.

2.1 Searching for Data Using the Web Interface

To begin obtaining this data, we must first access the Mikulski Archive for Space Telescopes (MAST) archive where we will do a search for UGC 2698, all IR, and all F160W observations. After submitting our request, we are taken the the *HST Search Results* page where we can see 3 options of data. To obtain the data needed for this tutorial, we must select the two data-sets with apertures of “IR" only and then select the “*Submit marked data for retrieval from STDADS*" button. Then, after filling out our email information, we must have only the *Uncalibrated* box checked in the *File Options* tab, before submitting the retrieval request to ST-DADS.

2.2 Obtaining Reference Files

We have written an automated code to acquire the necessary reference files for UGC 2698 F160W full and sub frame observations which can be seen below.

In [3]:
```
#import modules
from astroquery.mast import Observations
import os
import shutil
import subprocess
import glob

Type = ['full','sub']

for i in range(len(Type)):
    #change to the directory with the raw fits file
    os.chdir('/home/student/hst_compactgals/ugc2698/f160w_'+Type[i]+'_raw/')

    #set some environment variables
    os.environ['CRDS_SERVER_URL'] = 'https://hst-crds.stsci.edu'
    os.environ['CRDS_PATH'] = os.path.abspath(os.path.join('.', 'reference_files') )

    #call the CRDS tool to find the best reference files. this will take some time to run.
    subprocess.check_output('crds bestrefs --files *_raw.fits --sync-references=1 --update-bestrefs',
                            shell=True,stderr=subprocess.DEVNULL)


    #clean up. move the fits files to reference_files/ and remove everything else .
    for file in glob.glob ((os.path.join ('.', 'reference_files', 'references','hst', 'wfc3') ) +'/*.fits'):
        os.rename(file,(os.path.join ('.', 'reference_files')) + '/' + \
                  os.path.basename(file))

    shutil.rmtree(os.path.join ('.','reference_files','config'))
    shutil.rmtree(os.path.join ('.','reference_files','mappings'))
    shutil.rmtree(os.path.join ('.','reference_files','references'))
```

Figure 1: Our automated code in obtaining our reference files.

The output of this code is listed below for both the full and sub frame. The full frame output is listed above, while the sub frame output is listed below.

```
(astroconda) student@ubuntu:~/hst_compactgals/ugc2698/f160w_full_raw/reference_files$ ls
3562021pi_mdz.fits  35620330i_bpx.fits  4ac1921li_pfl.fits  q911321mi_osc.fits  u1k1727mi_lin.fits  w3m18525i_idc.fits
3562021si_drk.fits  4ac18260i_dfl.fits  4af1533ai_imp.fits  t2c16200i_ccd.fits  u6a1748ri_crr.fits
(astroconda) student@ubuntu:~/hst_compactgals/ugc2698/f160w_full_raw/reference_files$ ls ~/hst_compactgals/ugc2698/f160w_sub_raw/refer
ence_files/
3562016qi_drk.fits  35620330i_bpx.fits  4ac1921li_pfl.fits  q911321mi_osc.fits  u1k1727mi_lin.fits  w3m18525i_idc.fits
3562021pi_mdz.fits  4ac18260i_dfl.fits  4af1533ai_imp.fits  t2c16200i_ccd.fits  u6a1748ri_crr.fits
(astroconda) student@ubuntu:~/hst_compactgals/ugc2698/f160w_full_raw/reference_files$
```

Figure 2: The reference files listed, the full frame is above, the sub frame is below.

2.3 Initial Processing with CALWF3

After running the the raw.fits files through the calwf3 pipeline, the output images are listed as flt.fits files. These files differ from the raw images in that there is a decreased contrast in the flt files than the raw files. Furthermore, calwf3 seems to “smoothen" the sharp edges between the four quadrants in the raw files, resulting in the flt files to not appear divided, and more like one full image.

2.4 Completing the Data Reduction with AstroDrizzle

Before we drizzle the images, we need to ensure they are properly aligned (or registered). To do this we must use DS9, an image viewer, to examine the brightest pixel in UGC 2698, and record the right ascension (RA) and declination (Dec). We have our results listed in Table 1 below.

Table 1. Center of UGC 2698

File name	Right Ascension	Declination
Full Frame		
ic7011r2q_flt.fits	3:22:02.9067	+40:51:50.017
ic7011r3q_flt.fits	3:22:02.9018	+40:51:50.006
ic7011r5q_flt.fits	3:22:02.9072	+40:51:50.035
Sub Frame		
ic7011r7q_flt.fits	3:22:02.9104	+40:51:50.060
ic7011r8q_flt.fits	3:22:02.9148	+40:51:50.043
ic7011r9q_flt.fits	3:22:02.9093	+40:51:49.985
ic7011raq_flt.fits	3:22:02.9132	+40:51:49.991

Note. — The first 3 files belong in the full frame while the last 4 files belong in the sub frame.

As seen in the table above, we can take the RA and Dec of UGC 2698 to be 3:22:02.9071, +40:51:50.035 for the full frame and 3:22:02.9148, +40:51:50.043 for the sub frame. Since all the RA and Dec values are within 0.1 arcsecond of each other, we can assume that the F160W images are registered pointing the difference in measurements to be possible human error.

What me must do next is run both sets of files through Astrodrizzle. To do this, we must find two parameters, pixfrac and pixscale, that would result in an image that has a high definition with no holes in it. For a short example in extremes, we will alter pixfrac between two extreme values, $1.0''$, and $0.01''$, while keeping our pixscale at a default value, $0.1283''$.

When doing this, we can see that the pixfrac value of $1.0''$ has no holes in the image for both the full and sub frame. But when using $0.01''$, the sub frame has holes occurring around the edges of the image. specifically, the top and bottom edge seem to have “nan" values that are roughly evenly spread out from another, while the right and left edges seem to only have the “nan" values occur closer to the corner of the image. To clearly see what is going on here, we must observe the full frame as well. The full frame also has these “nan" values occurring throughout the image, yet they seem to fall within a pattern, suggesting the holes are the errors that occur in taking a 3D object and plotting it on a 2D plane while giving each pixel the same value. Not much different than when plotting a sphere on a globe, the holes that occur in our image are the creases where the sky begins to bend.

Now, to find a value for pixscale that would result in a high resolution image, we can run this experiment again, this time keeping pixfrac constant at $1.0''$, while adjusting pixscale in the range of $0.11''$ to $0.05''$ in decreasing steps of $0.01''$. We chose these values as we wanted to fall below the default measurement for pixscale ($0.1283''$), while also staying above any value below $0.05''$ as generally the pixscale parameter is never held at a value less than $\sim 0.4-0.5\times$ that of the default measurement. To check the effectiveness of these changes, we will be looking at Star 1 (highlighted in the instructions) and the contours around it.

Table 2. Differences in Pixscale values

Pixfrac	1.0″	1.0″	1.0″	1.0″	1.0″	1.0″	1.0″	1.0″
Pixscale	0.1283″	0.11″	0.10″	0.09″	0.08″	0.07″	0.06″	0.05″

When running through the differing pixscale values, a rough trend can be seen. If we hold the contour parameters to be the same throughout each image, it can be seen that as the pixscale lowers, the more “boxy" the contour lines begin to appear, with a strange peak model at a pixscale of 0.07″, which is shown below. Any pixscales .05″ and lower have uneven and warped contours, widening out from an ideal circular shape to a more rectangular shape, as seen with a pixscale value of 0.05″ also shown below.

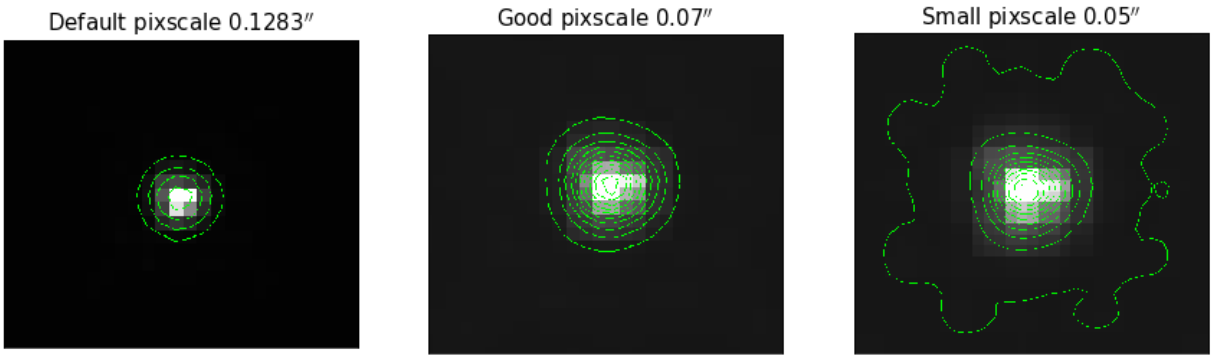

Figure 3: The 3 examples of our pixscale in the full frame, on the left we have our default value of 0.1283″, in the middle we have our good pixscale value of 0.07″, and on the right we have our small pixscale value of 0.05″

After deciding to go with a pixscale of 0.07″ due to its uniform and circular contours, we must now alter the pixfrac while keeping our pixscale constant. We will start this with a pixfrac of 1.0″ and fall in steps of 0.1″ until we reach a pixfrac of 0.0″. For each iteration, we will find the Full Width Half Max (FWHM) of star 1 while also attempting to find the standard deviation/median of a region around UGC 2698 using the weighted image. A rough guide is to keep the standard deviation/median below ~ 0.2 The region we chose to survey is a circle centered on 3:22:02.03572, + 40:52:18.914, with a radius of 0.003°.

When surveying each iteration, a clear trend is seen, as the pixfrac lowers, the resolution becomes worse and we get a plethora of “nan" values as seen in our example done previously in this section where we kept the pixscale at the default value while lowering the pixfrac to 0.01″. This appears to begin around a pixfrac of 0.4″ and as such it is hard to get accurate measurements for these values onward due to the “nan" pixels spreading throughout the image.

Table 3. Differences in Pixfrac values for the Full Frame

Pixfrac	Pixscale	FWHM	Standard Deviation/Median
1.0″	0.07″	2.53	0.059
0.9″	0.07″	2.47	0.127
0.8″	0.07″	2.46	0.214
0.7″	0.07″	2.46	0.325
0.6″	0.07″	2.40	0.454
0.5″	0.07″	2.43	0.592
0.4″	0.07″	2.48	0.721
0.3″	0.07″	2.64	0.856
0.2″	0.07″	2.92	0.953
0.1″	0.07″	1.61	0.919
0.0″	0.07″	1.54	0.887

After observing through each of the pixfrac values, we decided to choose the pixfrac value of 0.9″ and a pixscale value of 0.07″ as our final full frame science image. We chose our pixscale value of 0.07″ due to the sharp contours of Star 1 as seen in Figure 3. Furthermore, when looking through each of the pixfrac values while using the pixscale of 0.07″, we chose 0.9″ as our final. We did this due to the standard deviation/median of UGC 2698 being 0.127, well below 0.2, along with the FWHM of Star 1 being 2.47, close to the median and mode of the first 6 iterations. We only chose to observe the first 6 iterations as any image with a pixfrac of 0.4″ or lower were corrupted with “nan" values.

(1) An image of our final dirzzled image is generated below.

Figure 4: Our Final Full-Frame Science image of UGC 2698. This image has a pixfrac of 0.9″ and a pixscale of 0.07″.

2.5 Astrodrizzle cont.

While attempting to find the Pixfrac and Pixscale for the sub frame, we will be using the same iterations through Pixscale as Table 2. When comparing to the full frame, a similar trend can be seen, that is, when the pixscale falls below a certain threshold, the outer contour lines begin to warp. However, these contour lines differ from that of the full frame as they begin to form lopsided elliptical shapes, as apposed to the “boxy" shapes that we discussed previously. Below, are some examples of a good pixscale, small pixscale, and our default pixscale (0.1283′).

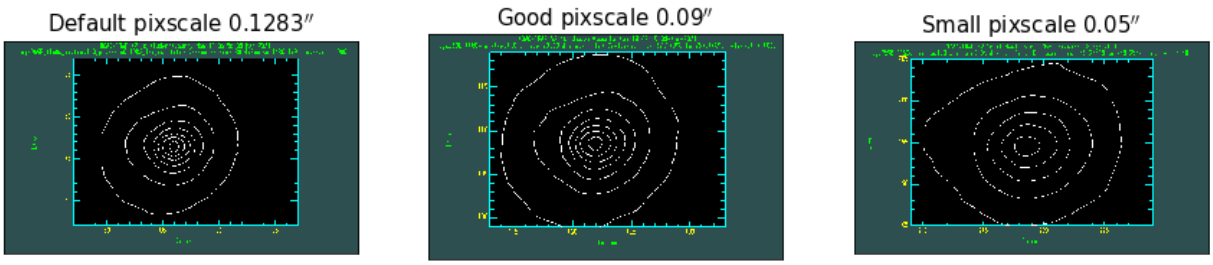

Figure 5: The 3 examples of our pixscale in the sub frame, on the left we have our default value of 0.1283″, in the middle we have our good pixscale value of 0.09″, and on the right we have our small pixscale value of 0.05″

Now, conducting the same process to find a reasonable pixfrac for the sub frame as we did in the full frame, our results are listed in Table 4 on the next page.

Due to our results, we chose our final pixfrac to be 1.0″ and our final pixscale to be 0.09″. This is due to this data having the smallest Standard Deviation/Median ratio in our sample. Strangely, a result of this data means we cannot use a smaller pixscale value when drizzing the sub array of F160W images when com pairing to the F160W full array images. We believe this is due to the image already at a smaller scale, if we were to shink the pixel size further, it would only cause further issues when taking data.

Our resulting final sub frame image is listed below.

Figure 6: Our Final Sub Frame Science image of UGC 2698. This image has a pixfrac of 1.0″ and a pixscale of 0.09″.

Table 4. Differences in Pixscale values for the Sub Frame

Pixfrac	Pixscale	FWHM	Standard Deviation/Median
1.0″	0.09″	5.38	0.287
0.9″	0.09″	5.32	0.431
0.8″	0.09″	5.23	0.377
0.7″	0.09″	5.13	0.381
0.6″	0.09″	5.04	0.385
0.5″	0.09″	4.98	0.453
0.4″	0.09″	4.90	0.355
0.3″	0.09″	4.78	0.340
0.2″	0.09″	4.70	0.385
0.1″	0.09″	4.66	0.406
0.0″	0.09″	4.60	0.419

2.6 Aperture and Surface Photometry

After examining the mask against the science image, we can say that the mask image is attempting to exclude all the light noise from nearby stars and other objects. The code we used to measure Star 1's magnitude is on the next page.

```
# import modules
from photutils import centroid_sources, CircularAperture
from photutils import CircularAnnulus, aperture_photometry
from astropy.stats import sigma_clipped_stats
import numpy as np
from astropy.io import fits

sci_img = fits.open('UGC2698_F160W_drz_sci.fits')[0].data
star_x, star_y = 2085, 1209      #Rough guesses for the position of the star
r = 16.6666666666667             #1 arcsecond in pixels
r_in = 2*r
r_out = 3*r
zero_point = 24.6949

def magnitude(apeture_phot, zero_point):
    m = -2.5*np.log10(apeture_phot) + zero_point
    return(m)

# start by finding the star's center.
xcen, ycen = centroid_sources(sci_img, star_x, star_y, box_size=11)

# define the circular aperture and sky annulus.
aperture = CircularAperture((xcen[0],ycen[0]),r)
annulus_aperture = CircularAnnulus((xcen[0],ycen[0]),r_in,r_out)

# determine the sum of the flux within the aperture and annulus
phot_table_star = aperture_photometry(sci_img,aperture)
phot_table_sky = aperture_photometry(sci_img,annulus_aperture)
print('The total number of counts in the circular apeture before background subtraction is: '\
      +str(round(phot_table_star['aperture_sum'][0],3)))

# instead of determining the straight average, we will determine the mean sky
# value after sigma-clipping in case there are bad pixels within our sky annulus.

# create a mask defining the sky annulus region, and applying that onto
# our science image, turning it into a 1D array
annulus_masks = annulus_aperture.to_mask(method='center')
annulus_data = annulus_masks.multiply(sci_img)
annulus_data_1d = annulus_data[annulus_masks.data > 0]
print('The total number of counnts in the sky annulus prior to sigma-clipping is: '\
      +str(round(np.sum(annulus_data_1d),4)))

# Gettting the mean, median, and standard deviationv of the sigma-clipped annulus
mean_sigclip, median_sigclip, stdev_sigclip = \
    sigma_clipped_stats(annulus_data_1d, sigma = 3, maxiters = 5)
print('The average sky value in the annulus after sigma-clipping is: '\
      +str(round(mean_sigclip,3)))

# multiply by the aperture area to get the background value within the circular aperture.
background = median_sigclip * aperture.area
print('The background value within the circular apeture is: '+str(round(background,3)))

# Take the counts within the circular aperture and subtract the background value.
final_result = phot_table_star['aperture_sum'] - background
print('The final sky-subtracted counts corresponding to the star within the ciruclar apeture is:
 '\
     +str(round(final_result[0],3)))

# Finding the magnitude of our star by using the magnitude function defined above
mag = magnitude(final_result[0],zero_point)
print('Star 1\'s apparent magnitude is: '+str(round(mag,4)))
```

Figure 7: The code we used to find Star 1's magnitude and other important attributes.

The output to the code in Figure 7 is listed below.

```
(astroconda) student@ubuntu:~/hst_compactgals/ugc2698/photometry$ python Apeture_example.py
The total number of counts in the circular apeture before background subtraction is: 10717189.994
The total number of counnts in the sky annulus prior to sigma-clipping is: 352015.5615
The average sky value in the annulus after sigma-clipping is: 69.505
The background value within the circular apeture is: 58738.222
The final sky-subtracted counts corresponding to the star within the ciruclar apeture is: 10658451.773
Star 1's apparent magnitude is: 7.1257
(astroconda) student@ubuntu:~/hst_compactgals/ugc2698/photometry$
```

Figure 8: The output to Figure 7's code.

Figure 8 lists that the total number of counts in the circular aperture prior to background subtraction is: 10,717,189.994, the total number of counts in the sky annulus prior to the sigma-clipping algorithm is: 352,015.561, the average sky value in the annulus after sigma-clipping is: 69.505, the background value within the circular aperture is: 58738.22, the final sky-subtracted counts corresponding to the star within the circular aperture is: 10,658,541.773, and Star 1's apparent magnitude is 7.1257.

When executing the code in Figure 7, we did not subtract the total number of counts in the sky annulus from the total number of counts within the circular aperture due to the result not successfully removing the background value. To successfully remove the background value, we must first find an area around the star that has no light pollution from another source and remove pixels that are extreme outliers using sigma-clipping, finding the average values of said remaining pixels. We must then multiply this value by the area of the region we chose, this will result in the average background value we desire.

Below is the code we wrote to determine UGC 2698's surface brightness in units of mag arcsec^{-2}.

```
# import modules
from photutils.isophote import EllipseGeometry, Ellipse
import numpy.ma as ma
import matplotlib.pyplot as plt
from astropy.io import fits
import numpy as np

sci_img = fits.open('UGC2698_F160W_drz_sci.fits')[0].data
mask_img_in = fits.open('UGC2698_F160W_drz_mask.fits')[0].data
x_init = 1469
y_init = 1405
sma_init = 4
eps_init = .3333333
pa_init_rad = 6.17
zero_point = 24.6949



# set an initial ellipse for measuring the first isophote. need to supply
# starting guesses for the isophote parameters: the x and y center (in pixels),
# the semi-major axis (in pixels), the ellipticity, and position angle (in
# radians).
ellipse_geo = EllipseGeometry(x0=x_init, y0=y_init, sma=sma_init, eps=eps_init, pa=pa_init_rad)

# apply a mask (mask_img_in) to the science image (sci_img)
sci_img_masked = ma.masked_array(sci_img, mask=mask_img_in)

# fit isophotes to the image.
ellipse = Ellipse(sci_img_masked, ellipse_geo)
isolist = ellipse.fit_image()

# isolist includes attributes like intens (the mean intensity value along the
# elliptical path), pa (the position angle in radians of the ellipse), eps (the
# ellipticity of the ellipse), and sma (the semi-major axis length in pixels).
# as an example, print each isophotal intensity.
print(isolist.intens)

# as another example, plot the masked science image and overplot the isophotes.
# this is a quick plot for the purposes of exploring the data and doesn't have
# the labels, units, colorbar, tick marks, etc. that a plot should have.
fig, ax = plt.subplots(figsize=(8, 8))
ax.imshow(sci_img_masked, vmin=0, vmax=1200)
# don't show the full image, but zoom in a little
ax.set_xlim([600, 2400])
ax.set_ylim([500, 2500])
# overplot the isophotes in white
smas_plot = np.linspace(0, 500, 20)
for sma in smas_plot:
    iso = isolist.get_closest(sma)
    x, y, = iso.sampled_coordinates()
    ax.plot(x, y, color='white')
# save the plot to a png file
plt.show()
plt.savefig('showellipses_u2698.png',bbox_image='tight')
plt.close()
```

Figure 9: Galaxy Surface Brightness Code.

We then proceeded to plot the surface brightness, position angle, and ellipticity as functions of the semi-major axis shown on the next page.

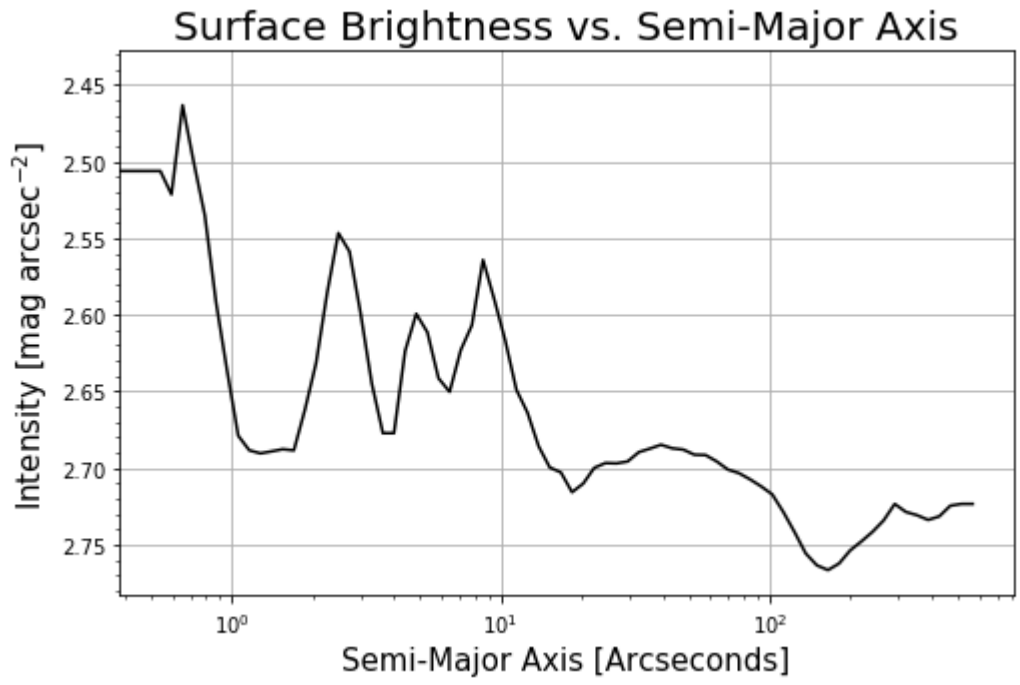

Figure 10: Surface Brightness as a function of the semi-major axis.

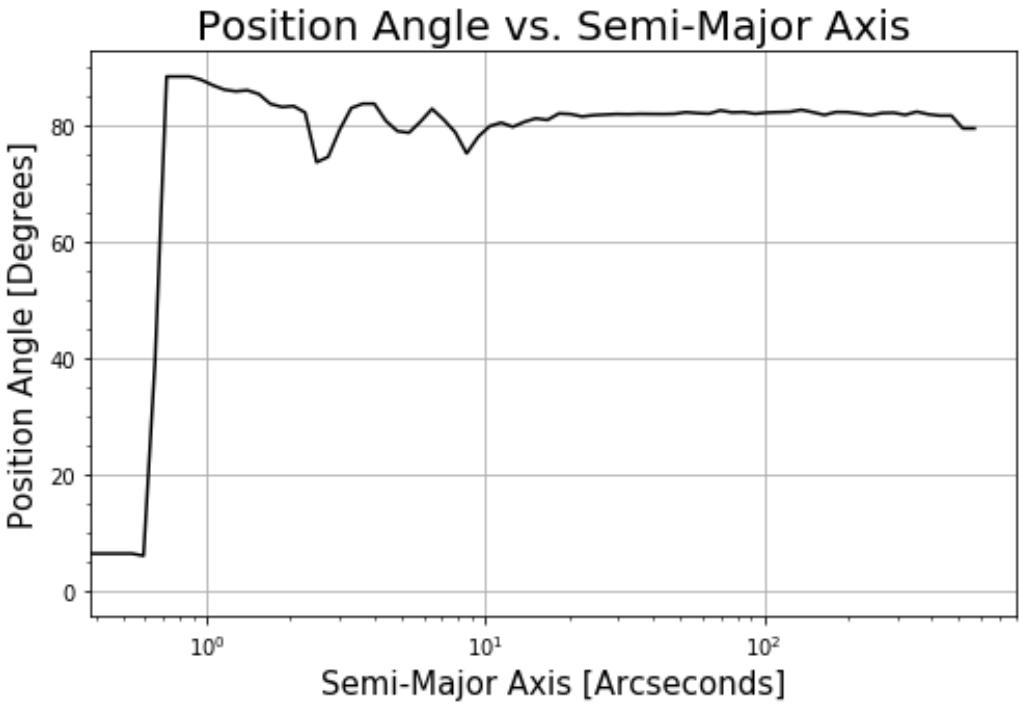

Figure 11: Position Angle as a function of the semi-major axis.

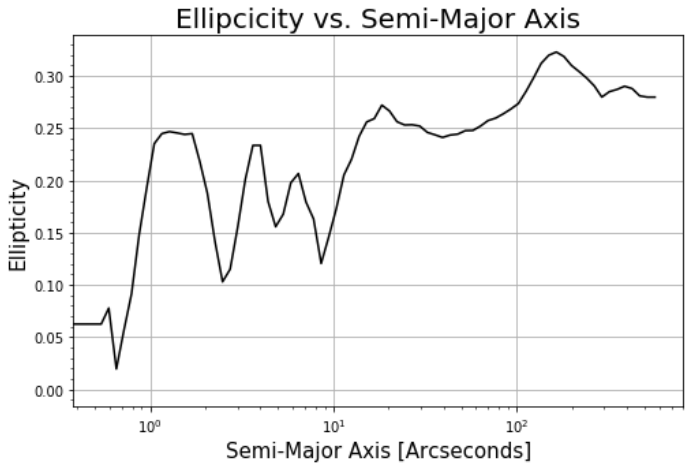

Figure 12: Ellipticity as a function of the semi-major axis.

Furthermore, we also wanted to create a model image of UGC 2698 and compare that to the masked science image to find the residual image. Figure 13 holds the code we used to do this and Figure 14 is the output of the model, residual, and masked science image.

```
In [33]: from photutils.isophote import build_ellipse_model
         import numpy as np

         #average pixel from regions ([900,1050],[900,1050])
         fill = np.mean(sci_img_masked[900:1050,900:1050])
         shape = np.shape(ellipse.image)

         #creating the model image
         model_img = build_ellipse_model(shape,isolist,fill)

         #creating the residual image
         residual_img = sci_img_masked-model_img
```

Figure 13: Code used to create the Model and Residual Images.

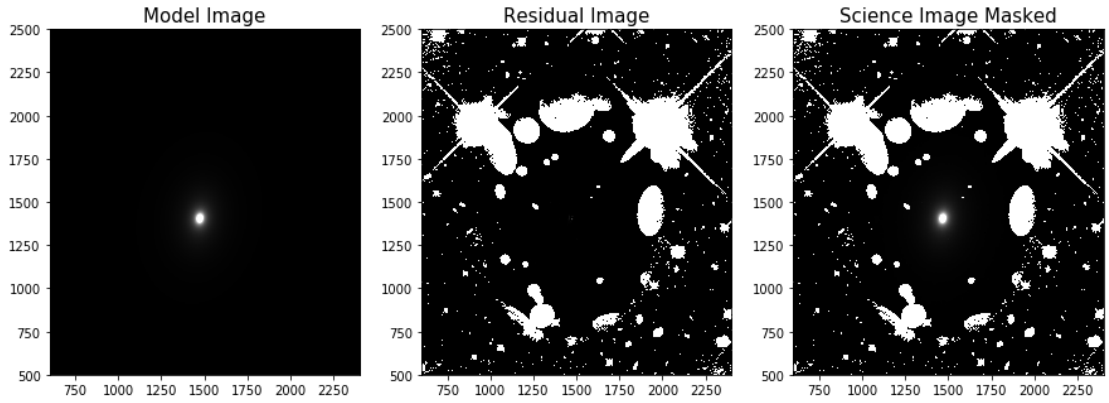

Figure 14: Our Model, Residual, and Masked Science Images.

2.7 Measuring Galaxy Sizes with Galfit

Below is a screen shot of our input file for Galfit.

```
================================================================================
# IMAGE and GALFIT CONTROL PARAMETERS
A) UGC2698_F160W_drz_sci.fits              # Input data image (FITS file)
B) UGC2698_F160W_galfit01.fits             # Output data image block
C) none                                    # Sigma image name (made from data if blank or "none")
D) f160w_psf.fits                          # Input PSF image and (optional) diffusion kernel
E) 1                                       # PSF fine sampling factor relative to data
F) UGC2698_F160W_drz_mask.fits             # Bad pixel mask (FITS image or ASCII coord list)
G) none                                    # File with parameter constraints (ASCII file)
H) 1  2710  1  2637                        # Image region to fit (xmin xmax ymin ymax)
I) 300      300                            # Size of the convolution box (x y)
J) 24.6949                                 # Magnitude photometric zeropoint
K) 0.06  0.06                              # 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

# INITIAL FITTING PARAMETERS
#
#   For object type, the allowed functions are:
#       nuker, sersic, expdisk, devauc, king, psf, gaussian, moffat,
#       ferrer, powersic, sky, and isophote.
#
#   Hidden parameters will only appear when they're specified:
#       C0 (diskyness/boxyness),
#       Fn (n=integer, Azimuthal Fourier Modes),
#       R0-R10 (PA rotation, for creating spiral structures).
#
# ------------------------------------------------------------------------------
#   par)    par value(s)    fit toggle(s)    # parameter description
# ------------------------------------------------------------------------------

# Object number: 1
 0) sersic           # object type
 1) 1469 1405  1 1       # position x, y        [pixel]
 3) 7.125       1       # total magnitude
 4) 26.5        1       #     R_e              [Pixels]
 5) 1       1       # Sersic exponent (deVauc=4, expdisk=1)
 9) 0.113       1       # axis ratio (b/a)
10) 6.17        1       # position angle (PA)  [Degrees: Up=0, Left=90]
 Z) 0                   #  Skip this model in output image?  (yes=1, no=0)

================================================================================
```

Figure 15: Our initial Parameters for our Galfit Input File.

The resulting fits file after using the parameters in figure 15 shared this result for our data, Best fit, and residual image.

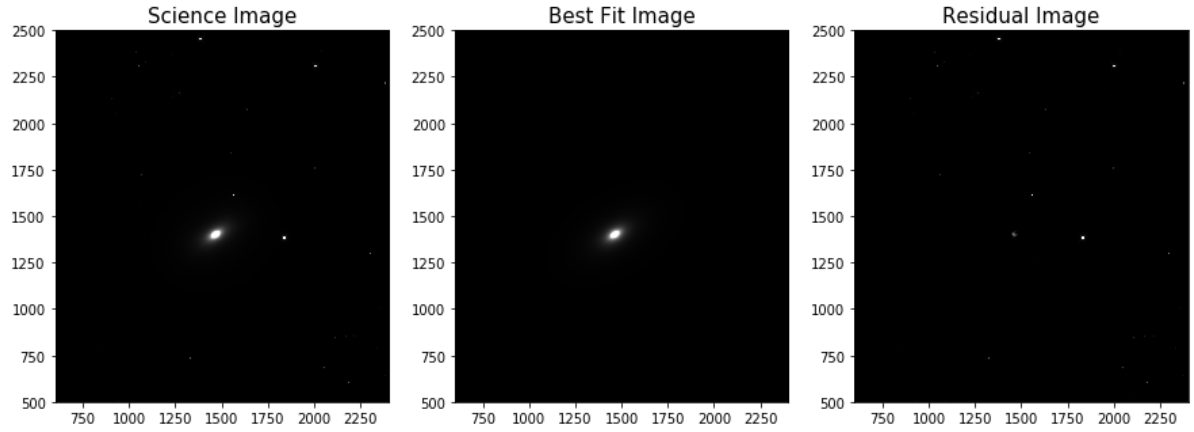

Figure 16: Our output files from Galfit.

Furthermore, the Sersic parameters we received for UGC 2698 follow below.

Table 5. Our Best-fit values of the Sersic Parameters for UGC 2698

Sersic Index	Data
X position (pixels)	1469.20
Y position (pixels)	1405.72
Total Magnitude	9.91
Effective Radius (pixels)	167.49
Sersic Exponent	5.09
Axis Ratio (b/a)	0.72
Position Angle (radians)	-8.11

Table 6. Translating the effective radius in Pixels to kpc to our galactic sample.

Galaxy	r_e in Pixels	r_e in KPC
MRK1216	106.84	1,255,191.89
NGC1270	118.94	1,429,143.39
NGC1271	98.63	1,429,509.74
NGC1281	117.99	1,788,819.57
NGC2767	131.54	1,822,605.48
PGC11179	76.05	1,022,598.65
PGC12562	51.92	956,628.78
PGC32873	76.82	944,538.96
PGC70520	90.29	1,421,173.28
UGC2698	167.49	1,643,584.82
UGC3816	193.5	3,022,951.24

2.8 Comparing to Other Massive Galaxies

When looking at 11 different galaxies, we wish to translate the effective radius r_e in pixels to kpc. We have done this above in Table 6.

Below is a chunk of code we wrote to convert r_e to kpc.

In [21]:
```
Mass = [11.34, 11.31, 11.06, 11.00, 11.12, 11.16, 10.74, 11.28, 10.95, 11.58, 10.96]
err_up = [0.11, 0.10, 0.07, 0.08, 0.09, 0.06, 0.10, 0.04, 0.10, 0.01, 0.06]
err_dwn = [0.01, 0.12, 0.07, 0.08, 0.08, 0.08 , 0.09, 0.04, 0.12, 0.03, 0.04]
r_e = [106.84, 118.94, 98.63, 117.99, 131.54, 76.05, 51.92, 76.86, 90.29, 167.49, 193.50]
dist = [94, 69, 80, 60, 74, 94, 67, 112, 72, 89, 51] #MPC

z_cuttoff = [0.25, 0.75, 1.25, 1.75, 2.25]
log_A = [0.60, 0.42, 0.22, 0.09, -0.05]
alpha = [0.75, 0.71, 0.76,0.76, 0.76]
```

In [44]:
```
def find_kpc(R_eff, A, alpha, M_star):
    M_sun = 1.989e30  #kg
    kpc = R_eff/(A*(M_star/(5e10 * M_sun))**alpha)
    return(kpc)

M_sun = 1.989e30  #kg
MPC_z = 0.000237 #1 MPC = 0.000237 redshift
M_stellar = []
kpc = []
z = []
bin_1, bin_2, bin_3, bin_4, bin_5 = [], [], [], [], []

for i in range(len(Mass)):
    M_stellar.append(np.exp(Mass[i])*M_sun) #Gives masses in kg
    z.append(dist[i]*MPC_z)

    if z[i] <= 0.25:
        bin_1.append(z[i])
        A = np.exp(log_A[0])
        alph = alpha[0]
        kpc.append(find_kpc(r_e[i], A, alph, M_stellar[i]))

    if(z[i] < 0.25 and z[i] >= 0.75):
        bin_2.append(z[i])
        A = np.exp(log_A[1])
        alph = alpha[1]
        kpc.append(find_kpc(r_e[i], A, alph, M_stellar[i]))

    if(z[i] < 0.75 and z[i] >= 1.75):
        bin_3.append(z[i])
        A = np.exp(log_A[2])
        alph = alpha[2]
        kpc.append(find_kpc(r_e[i], A, alph, M_stellar[i]))

    if(z[i] < 1.25 and z[i] >= 1.75):
        bin_4.append(z[i])
        A = np.exp(log_A[3])
        alph = alpha[3]
        kpc.append(find_kpc(r_e[i], A, alph, M_stellar[i]))

    if(z[i] < 1.25 and z[i] >= 2.25):
        bin_5.append(z[i])
        A = np.exp(log_A[4])
        alph = alpha[4]
        kpc.append(find_kpc(r_e[i], A, alph, M_stellar[i]))
```

Figure 17: Our code for translating r_e to kpc.

We also wanted to include a Mass-size relation plot with our sample of galaxies, which can be seen below.

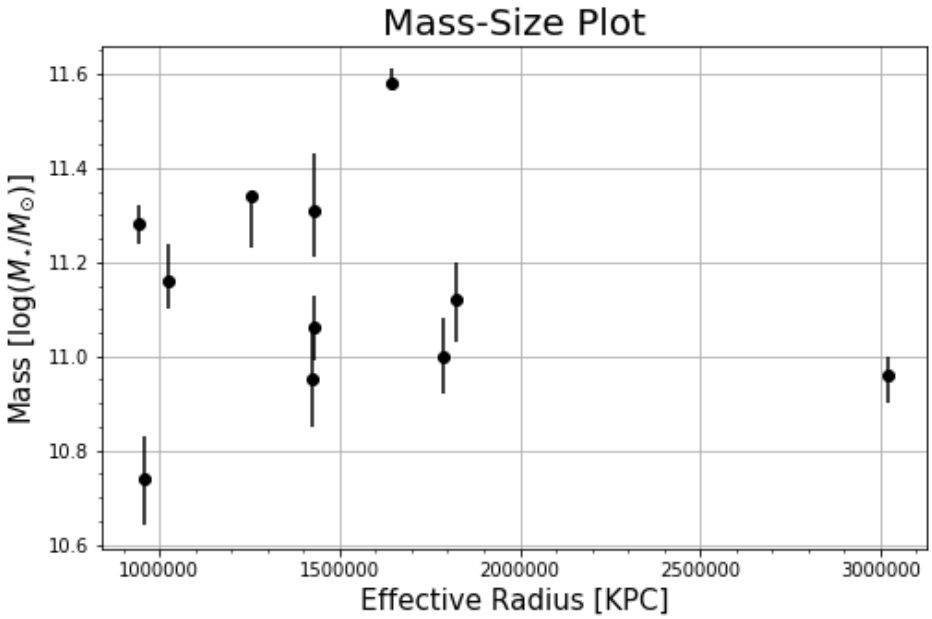

Figure 18: Our Mass-Size relation plot of the sample of galaxies

This plot generally follows the expected relationship that a larger the effective radius results in a more massive galaxy.