GroupModels

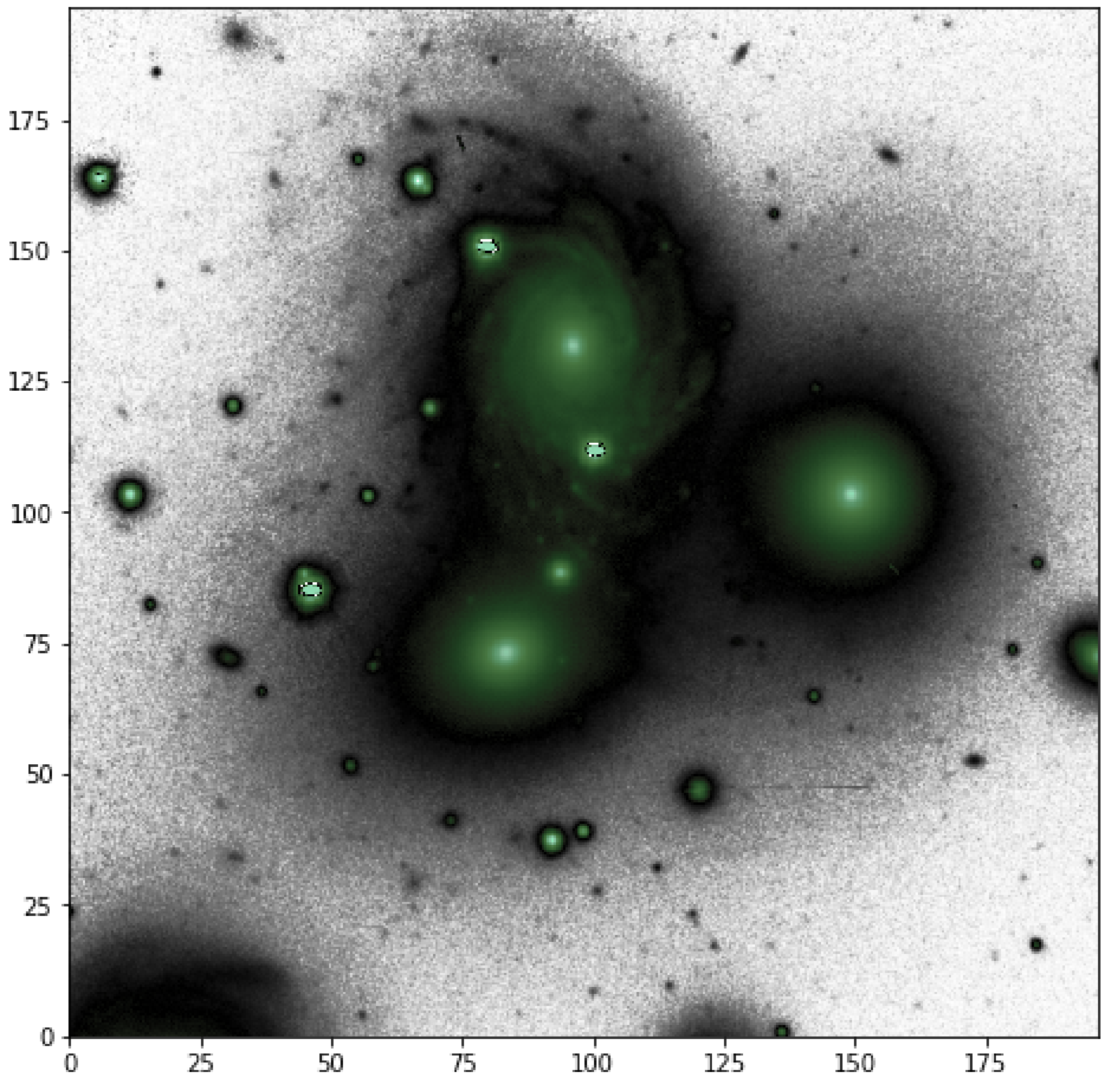
February 24, 2023

1 Group Models

Here you will learn how to combine models together into a larger, more complete, model of a given system. This is a powerful and necessary capability when analysing objects in crowded environments. As telescopes achieve ever deeper photometry we have learned that all environments are crowded when projected onto the sky!

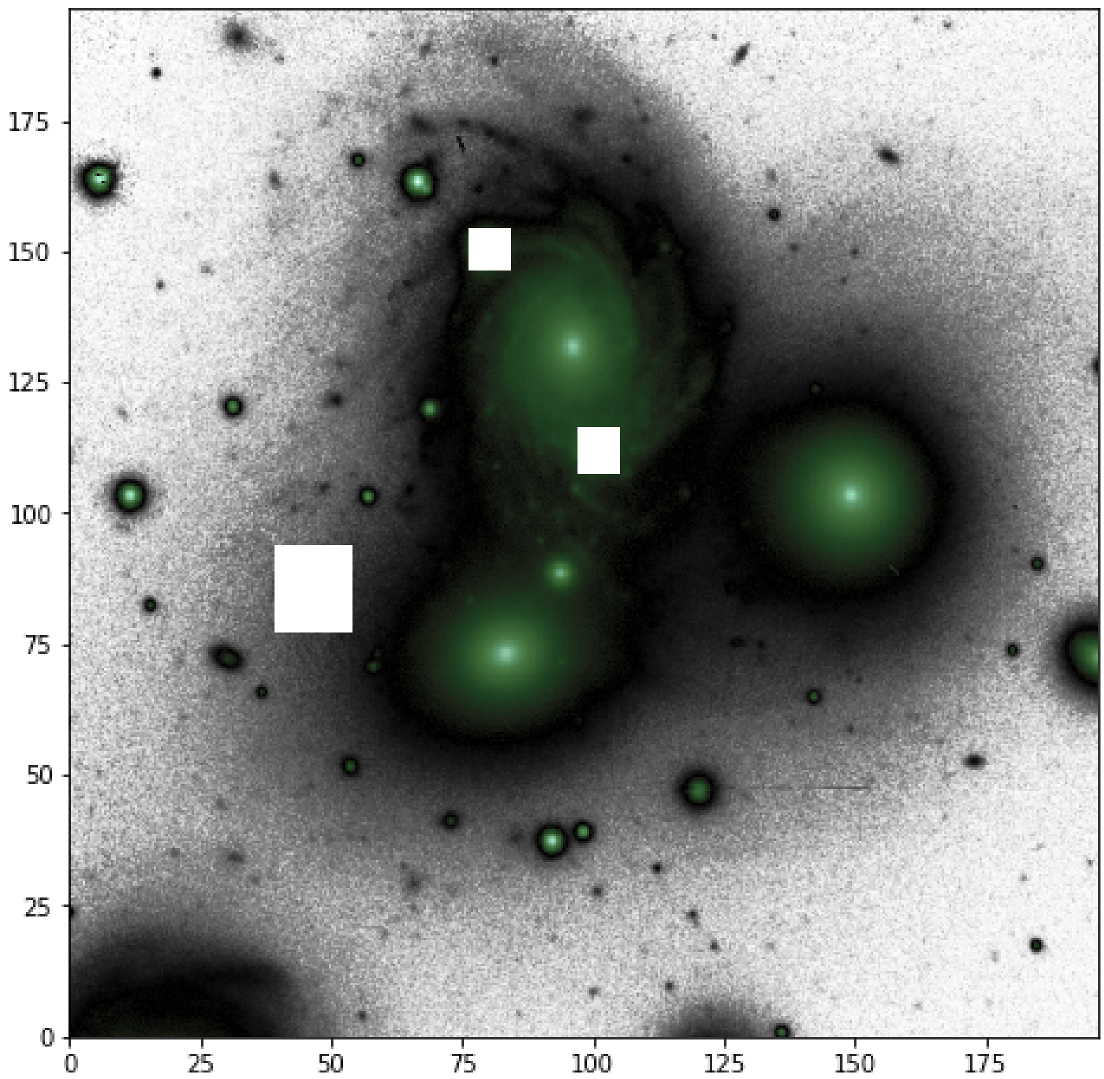
[1]:
```
import autoprof as ap
import numpy as np
import torch
from astropy.io import fits
import matplotlib.pyplot as plt
from scipy.stats import iqr
```

[2]:
```
# first let's download an image to play with
hdu = fits.open("https://www.legacysurvey.org/viewer/fits-cutout?ra=4.
↪5934&dec=30.0702&size=750&layer=ls-dr9&pixscale=0.262&bands=r")
target_data = np.array(hdu[0].data, dtype = np.float64)

target1 = ap.image.Target_Image(
    data = target_data,
    pixelscale = 0.262,
    zeropoint = 22.5,
)

fig1, ax1 = plt.subplots(figsize = (8,8))
ap.plots.target_image(fig1, ax1, target1)
plt.show()
```



[3]:
```
# We can see that there are some blown out stars in the image. There isn't much␣
↪that can be done with them except
# to mask them. A very careful modeller would only mask the blown out pixels␣
↪and then try to fit the rest, but
# today we are not very careful modellers.
mask = np.zeros(target_data.shape, dtype = bool)
mask[410:445,371:402] = True
mask[296:357 ,151:206] = True
mask[558:590,291:322] = True

pixelscale = 0.262
target2 = ap.image.Target_Image(
    data = target_data,
```

```
    pixelscale = pixelscale,
    zeropoint = 22.5,
    mask = mask, # now the target image has a mask of bad pixels
    variance = 0.001*np.abs(target_data + iqr(target_data,rng=[16,84])/2), # we␣
 ↪create a variance image, if the image is in counts then variance image =␣
 ↪image, in this case the sky has been subtracted so we add back in a certain␣
 ↪amount of variance
)

fig2, ax2 = plt.subplots(figsize = (8,8))
ap.plots.target_image(fig2, ax2, target2)
plt.show()
```

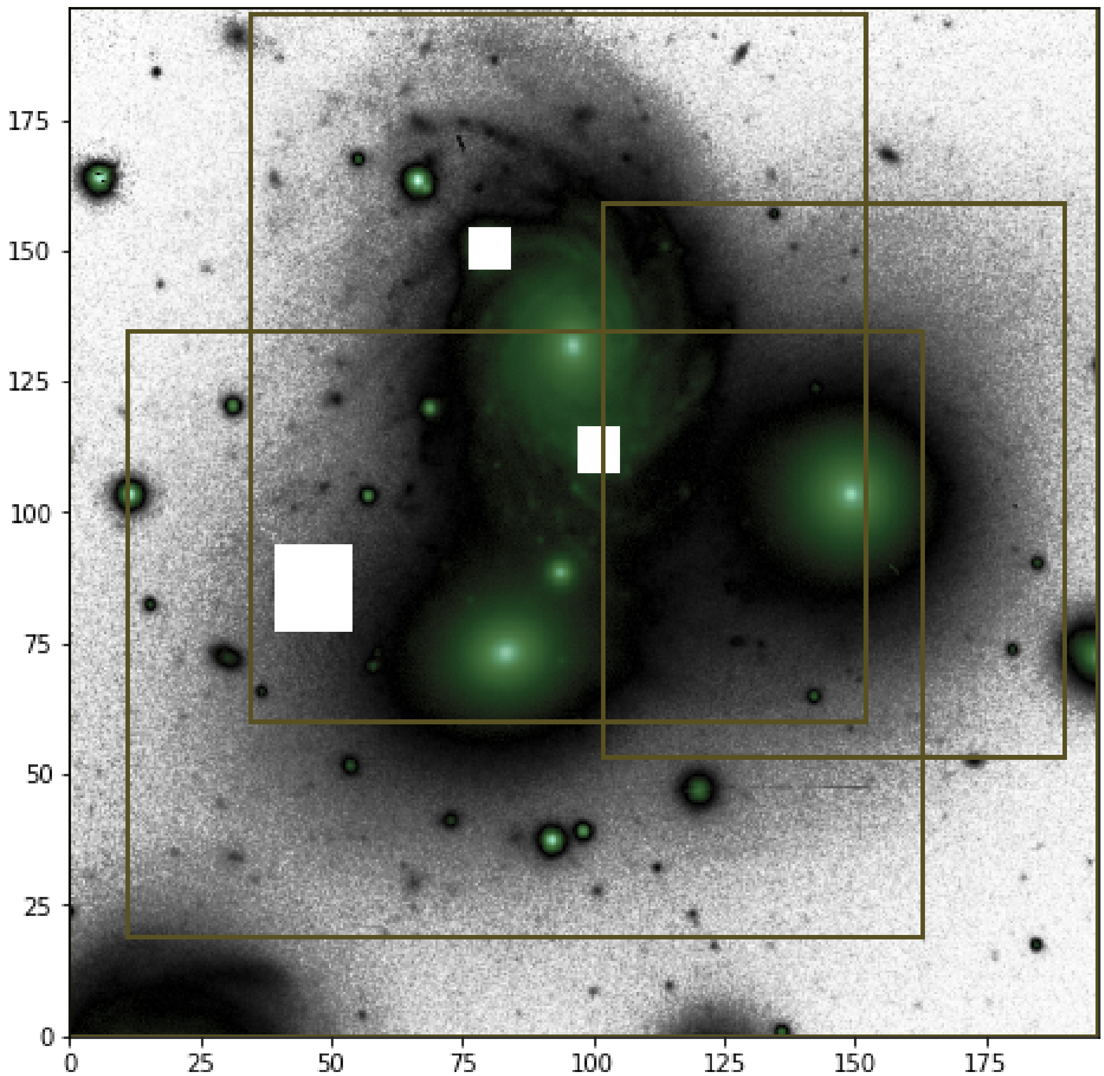


1.1 Group Model

A group model takes a list of other AutoProf_Model objects and tracks them such that they can be treated as a single larger model. When "initialize" is called on the group model, it simply calls "initialize" on all the individual models. The same is true for a number of other functions like finalize, sample, and so on. For fitting, however, the group model will collect the parameters from all the models together and pass them along as one group to the optimizer. When saving a group model, all the model states will be collected together into one large file.

The main difference when constructing a group model is that you must first create all the sub models that will go in it. Once constructed, a group model behaves just like any other model, in fact they are all built from the same base class.

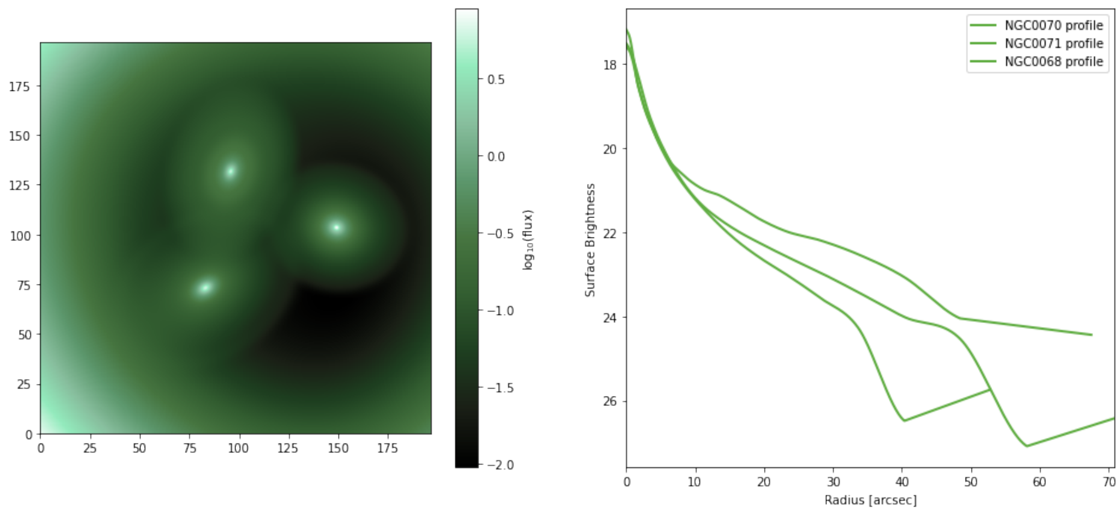
[4]:
```
# first we make the list of models to fit

# Note that we do not assign a target to these models at construction. This is␣
↪just a choice of style, it is possible
# to provide the target to each model separately if you wish. Note as well that␣
↪since a target isn't provided we need
# to give the windows in arcsec instead of pixels, to do this we provide the␣
↪window in the format (xmin,xmax,ymin,ymax)
model_kwargs = [
    {"name": "sky", "model_type": "flat sky model", "window": np.
↪array([0,750,0,750])*pixelscale},
    {"name": "NGC0070", "model_type": "nonparametric galaxy model", "window":␣
↪np.array([133,581,229,744])*pixelscale},
    {"name": "NGC0071", "model_type": "nonparametric galaxy model", "window":␣
↪np.array([43,622,72,513])*pixelscale},
    {"name": "NGC0068", "model_type": "nonparametric galaxy model", "window":␣
↪np.array([390,726,204,607])*pixelscale},
]

model_list = []
for M in model_kwargs:
    model_list.append(ap.models.AutoProf_Model(**M))

VV166Group = ap.models.AutoProf_Model(name = "VV166 Group", model_type = "group␣
↪model", model_list = model_list, target = target2)
VV166Group.sync_target()
fig3, ax3 = plt.subplots(figsize = (8,8))
ap.plots.target_image(fig3, ax3, VV166Group.target)
ap.plots.model_window(fig3, ax3, VV166Group)
plt.show()
```



[5]:
```python
# See if AutoProf can figure out starting parameters for these galaxies
VV166Group.initialize()

# The results are reasonable starting points, though far from a good model
fig4, ax4 = plt.subplots(1,2,figsize = (16,7))
ap.plots.model_image(fig4, ax4[0], VV166Group)
for M in VV166Group.model_list[1:]:
    ap.plots.galaxy_light_profile(fig4, ax4[1], M)
plt.legend()
plt.show()
```

```
initializing  VV166 Group
initializing  sky
initializing  NGC0070
```

```
initializing  NGC0071
initializing  NGC0068
```



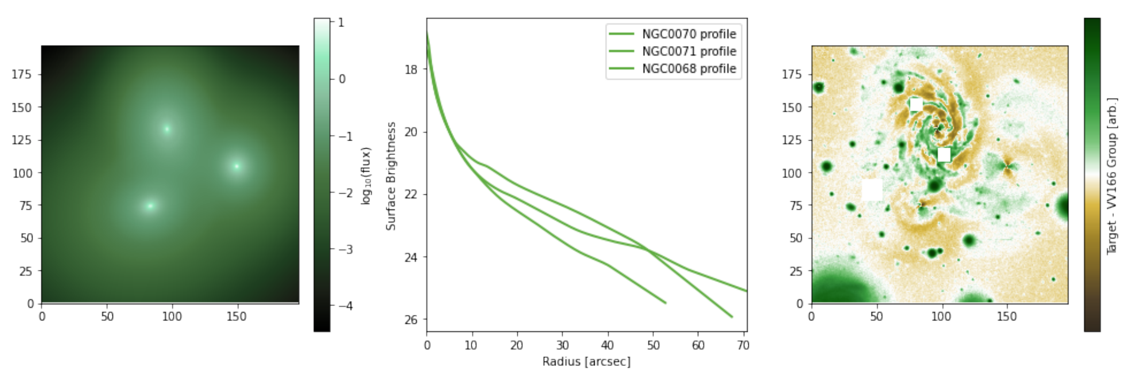
[6]:

```python
# Allow AutoProf to fit the target image with all 3 models simultaneously. In
↪total this is about 80 parameters!
result = ap.fit.LM(VV166Group, verbose = 1).fit()
print(result.message)
```

```
L: 1.0
---------init---------
L: 1.0
---------iter---------
LM loss: 7.605907471180069
accept
L: 0.1111111111111111
---------iter---------
reject
L: 1.2222222222222222
---------iter---------
LM loss: 7.088622957731792
accept
L: 0.13580246913580246
---------iter---------
LM loss: 6.695042440856809
accept
L: 0.015089163237311385
---------iter---------
LM loss: 6.594328649683231
accept
L: 0.0016765736930345982
```

```
---------iter---------
reject
L: 0.01844231062338058
---------iter---------
reject
L: 0.20286541685718637
---------iter---------
LM loss: 6.592284587869167
accept
L: 0.022540601873020708
---------iter---------
LM loss: 6.588786691864583
accept
L: 0.002504511319224523
---------iter---------
LM loss: 6.588741975423946
accept
L: 0.00027827903546939146
---------iter---------
LM loss: 6.588740085760209
accept
success
```

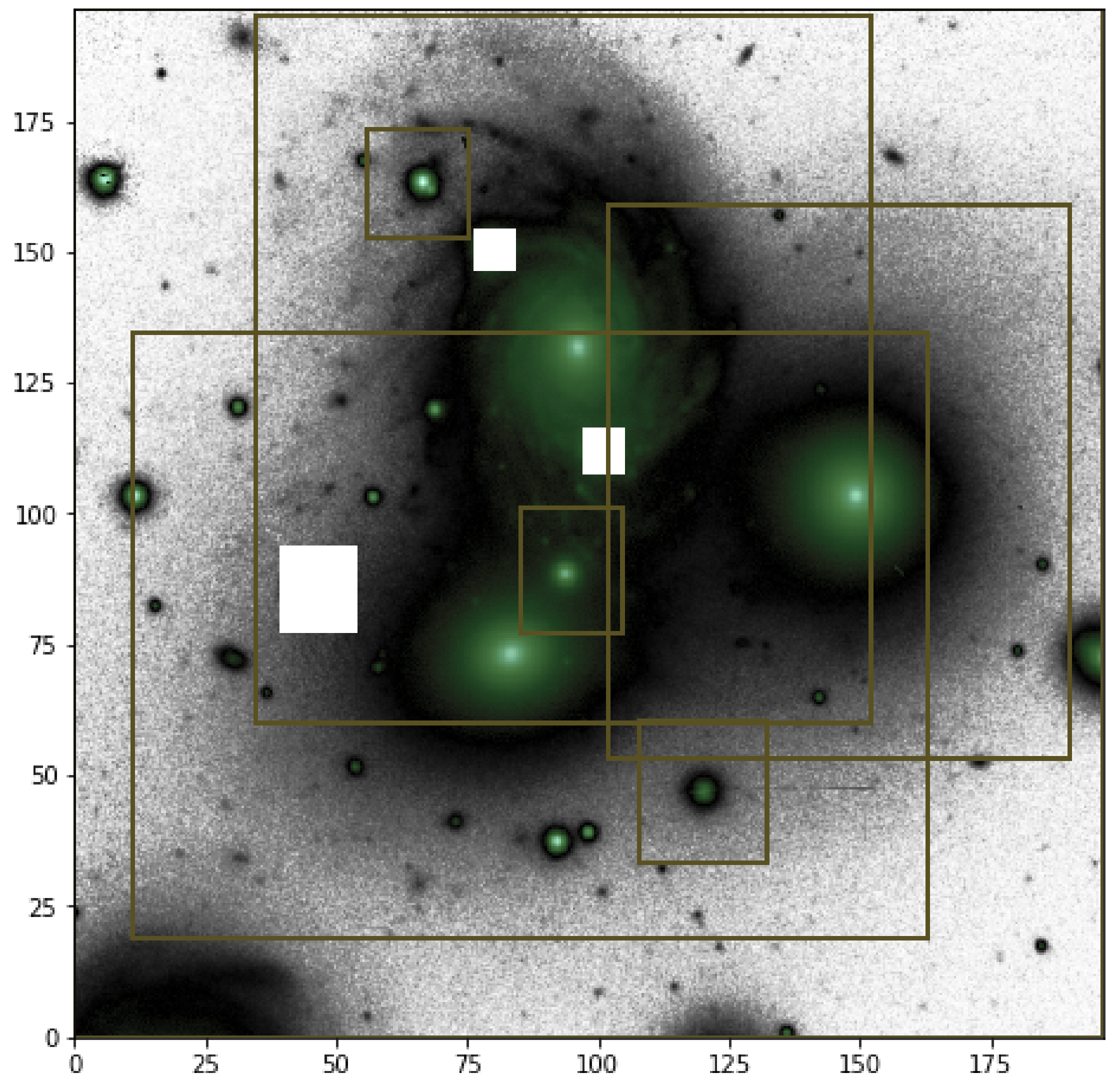
[7]:
```python
# Now we can see what the fitting has produced
fig5, ax5 = plt.subplots(1,3,figsize = (17,5))
ap.plots.model_image(fig5, ax5[0], VV166Group)
for M in VV166Group.model_list[1:]:
    ap.plots.galaxy_light_profile(fig5, ax5[1], M)
ax5[1].legend()
ap.plots.residual_image(fig5, ax5[2], VV166Group)
plt.show()
```



[8]:
```python
# The model will improve the more galaxies in the system we include
# By adding models now, we keep the fitted parameters from before.
```

```
VV166Group.add_model(ap.models.AutoProf_Model(name = "litte 1", model_type =␣
 ↪"sersic galaxy model", target = target2, window = [[325,400],[295,386]]))
VV166Group.add_model(ap.models.AutoProf_Model(name = "litte 2", model_type =␣
 ↪"sersic galaxy model", target = target2, window = [[412,504],[127,231]]))
VV166Group.add_model(ap.models.AutoProf_Model(name = "litte 3", model_type =␣
 ↪"sersic galaxy model", target = target2, window = [[214,288],[583,662]]))
```

[9]:
```
fig6, ax6 = plt.subplots(figsize = (8,8))
ap.plots.target_image(fig6, ax6, VV166Group.target)
ap.plots.model_window(fig6, ax6, VV166Group)
plt.show()
```



[10]:
```
# Initialize will only set parameter values for the new models, the old ones␣
↪will just be skipped
VV166Group.initialize()
```

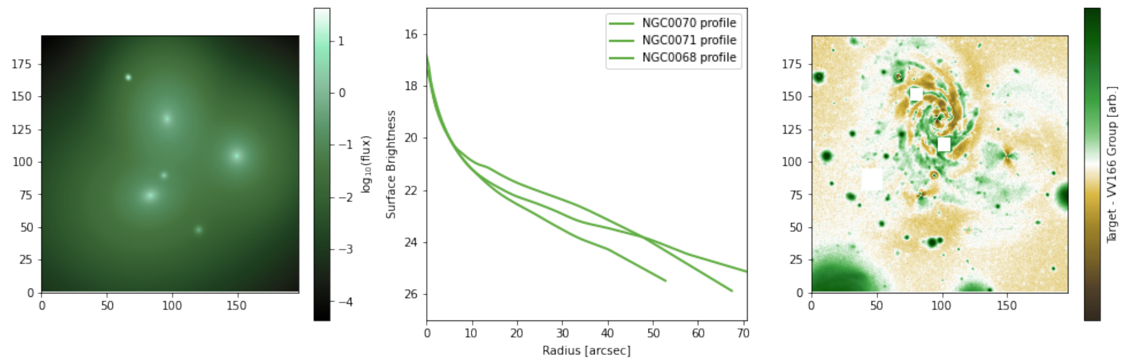
```
initializing  VV166 Group
initializing  sky
initializing  NGC0070
initializing  NGC0071
initializing  NGC0068
initializing  litte 1
initializing  litte 2
initializing  litte 3
```

[11]:
```
result = ap.fit.LM(VV166Group, verbose = 1).fit()
print(result.message)
```

```
L: 1.0
---------init---------
L: 1.0
---------iter---------
LM loss: 4.865957595337465
accept
L: 0.1111111111111111
---------iter---------
LM loss: 4.786247885999245
accept
L: 0.012345679012345678
---------iter---------
LM loss: 4.774774073461661
accept
L: 0.0013717421124828531
---------iter---------
LM loss: 4.773048132067801
accept
L: 0.00015241579027587256
---------iter---------
LM loss: 4.772959277914561
accept
L: 1.6935087808430286e-05
---------iter---------
LM loss: 4.772813911329844
accept
success
```

[12]:
```
# Now we can see what the fitting has produced
fig7, ax7 = plt.subplots(1,3,figsize = (17,5))
ap.plots.model_image(fig7, ax7[0], VV166Group)
```

```
# let's just plot the 3 main object profiles
for M in VV166Group.model_list[1:4]:
    ap.plots.galaxy_light_profile(fig7, ax7[1], M)
ax7[1].legend()
ax7[1].set_ylim([27,15])
ap.plots.residual_image(fig7, ax7[2], VV166Group)
plt.show()
```



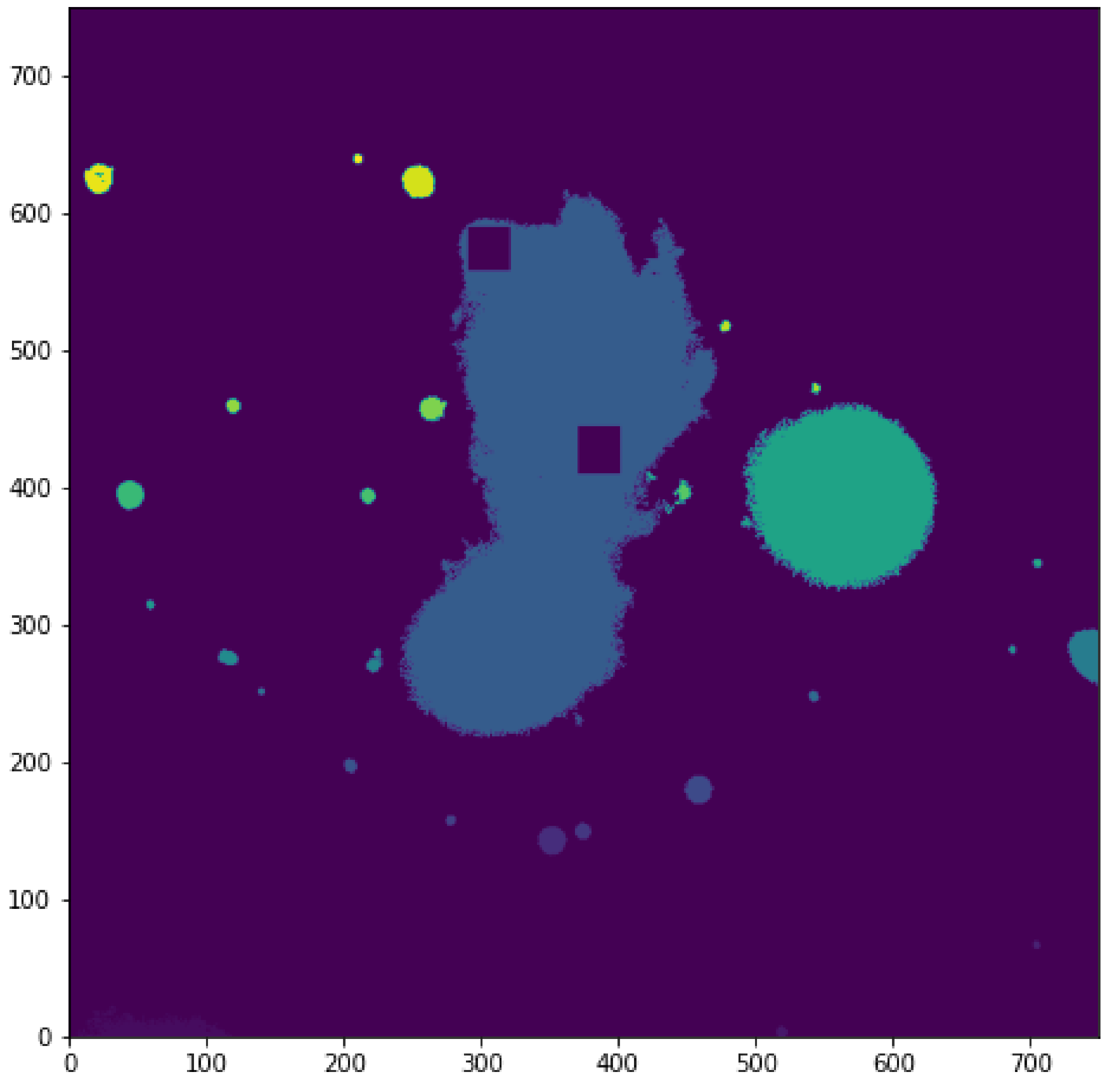
Which is even better than before. As more models are added, the fit should improve. In principle one could model eventually add models for every little smudge in the image. In practice, it is often better to just mask anything below a certain size.

1.2 Working with segmentation maps

A segmentation map provides information about the contents of an image. It gives the location and shape of any object which the algorithm was able to separate out and identify. This is exactly the information needed to construct the windows for a collection of AutoProf models.

Photutils provides an easy to use segmentation map implimentation so we use it here for simplicity. In many cases it may be required to use a more detailed segmentation map algorithm such as those implimented in Source Extractor and ProFound (among others), the principle is the same however since the end product for all of them has the same format.

```
[13]: from photutils.segmentation import detect_sources, deblend_sources
segmap = detect_sources(target_data, threshold = 0.1, npixels = 20, mask =␣
 ↪mask) # threshold and npixels determined just by playing around with the␣
 ↪values
fig8, ax8 = plt.subplots(figsize=(8,8))
ax8.imshow(segmap, origin = "lower")
plt.show()
```



[14]:
```
# This will convert the segmentation map into boxes that enclose the identified␣
↪pixels
windows = ap.utils.initialize.windows_from_segmentation_map(segmap.data)
# Next we filter out any segments which are too big, these are the NGC models␣
↪we already have set up
windows = ap.utils.initialize.filter_windows(windows, max_size = 100)
# Next we scale up the windows so that AutoProf can fit the faint parts of each␣
↪object as well
windows = ap.utils.initialize.scale_windows(windows, image_shape = target_data.
↪shape, expand_scale = 3, expand_border = 10)

del windows[20] # this is a segmented chunk of spiral arm, not a galaxy
del windows[23] # this is a segmented chunk of spiral arm, not a galaxy
```

```
del windows[24] # this is a segmented chunk of spiral arm, not a galaxy
del windows[28] # this is a segmented chunk of spiral arm, not a galaxy
del windows[29] # this is a repeat of little 2
del windows[7] # this is a repeat of little 3
print(windows)
```

```
{2: [[499, 540], [0, 24]], 3: [[689, 721], [49, 84]], 4: [[312, 392], [102,
182]], 5: [[348, 401], [123, 176]], 6: [[259, 297], [138, 176]], 8: [[183, 227],
[172, 222]], 10: [[352, 390], [209, 253]], 11: [[522, 563], [227, 268]], 12:
[[124, 156], [235, 267]], 13: [[697, 750], [206, 346]], 14: [[196, 249], [238,
309]], 15: [[85, 147], [251, 301]], 16: [[671, 703], [264, 299]], 17: [[42, 77],
[295, 333]], 19: [[688, 723], [327, 362]], 21: [[6, 83], [354, 434]], 22: [[193,
243], [367, 420]], 25: [[225, 305], [420, 494]], 26: [[96, 143], [434, 484]],
27: [[525, 563], [451, 492]], 30: [[0, 63], [583, 666]], 31: [[191, 229], [620,
658]]}
```

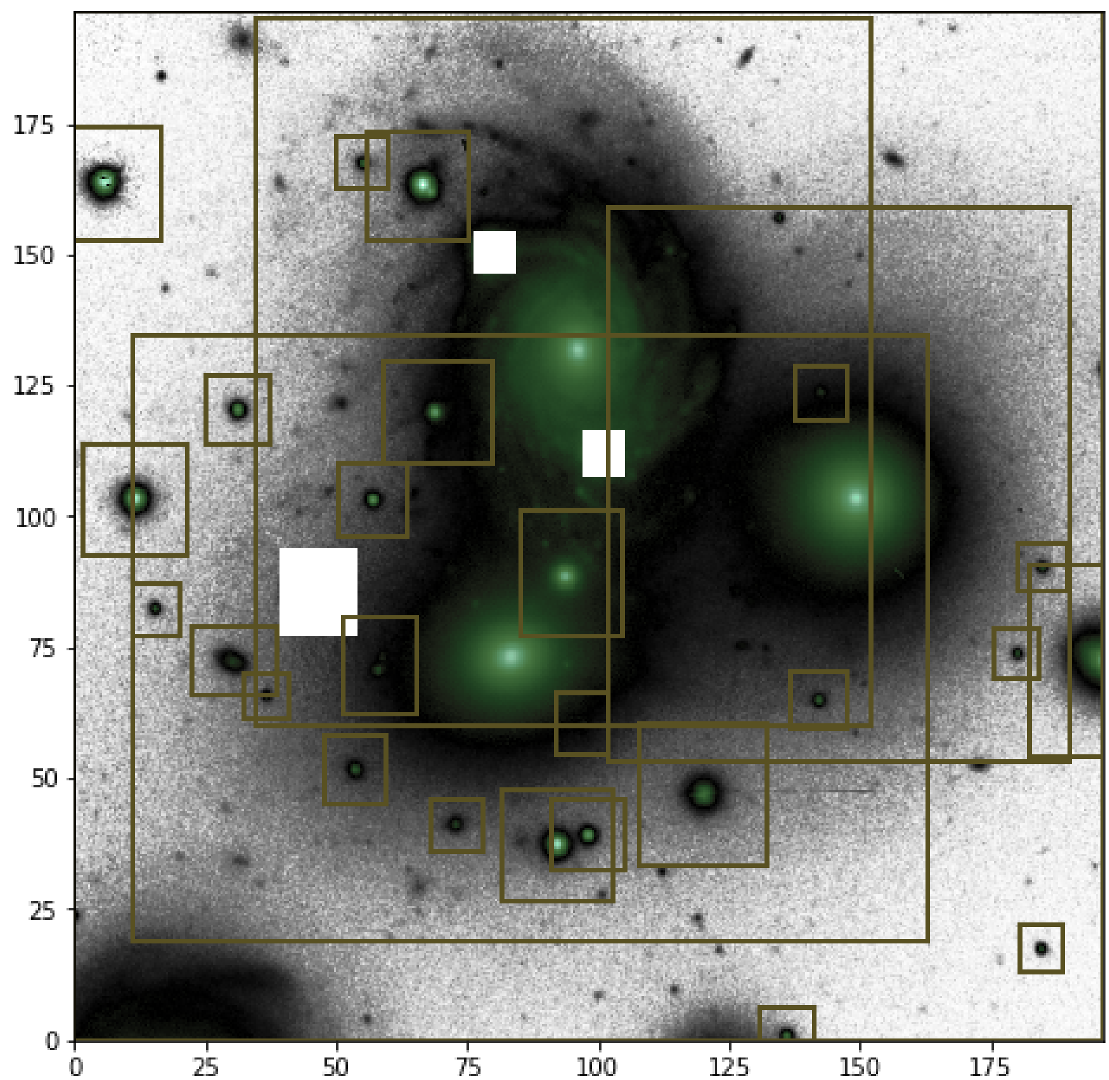
[15]:

```
# Now we use all the windows to add to the list of models
seg_models = []
for win in windows:
    seg_models.append({"name": f"minor object {win:02d}", "window":␣
↪windows[win], "model_type": "sersic galaxy model", "target": target2})

# we make a new set of models for simplicity
for M in seg_models:
    VV166Group.add_model(ap.models.AutoProf_Model(**M))

VV166Group.initialize()
```

```
initializing  VV166 Group
initializing  sky
initializing  NGC0070
initializing  NGC0071
initializing  NGC0068
initializing  litte 1
initializing  litte 2
initializing  litte 3
initializing  minor object 02
initializing  minor object 03
initializing  minor object 04
initializing  minor object 05
initializing  minor object 06
initializing  minor object 08
initializing  minor object 10
initializing  minor object 11
initializing  minor object 12
initializing  minor object 13
initializing  minor object 14
```

```
initializing  minor object 15
initializing  minor object 16
initializing  minor object 17
initializing  minor object 19
initializing  minor object 21
initializing  minor object 22
initializing  minor object 25
initializing  minor object 26
initializing  minor object 27
initializing  minor object 30
initializing  minor object 31
```

[16]:
```
fig9, ax9 = plt.subplots(figsize = (8,8))
ap.plots.target_image(fig9, ax9, VV166Group.target)
ap.plots.model_window(fig9, ax9, VV166Group)
plt.show()
```



[17]:
```
# This is now a very complex model composed of about 30 sub-models! In total␣
↪253 parameters! While it is
# possible for the AutoProf Levenberg-Marquardt (LM) algorithm to fully␣
↪optimize this model, it is faster in this
# case to apply an iterative fit. AutoProf will apply LM optimization one model␣
↪at a time and cycle through all
# the models until the results converge. See the tutorial on AutoProf fitting␣
↪for more details on the fit methods.
result = ap.fit.Iter(VV166Group, method = ap.fit.LM, verbose = 1).fit()
print(result.message)

# Other technqiues that can help for difficult fits:
# - Try running some gradient descent steps (maybe 100) before doing LM
# - Try changing the initial parameters. AutoProf seeks a local minimum so make␣
↪sure its the right one!
# - Fit the large models in the frame first, then add in the smaller ones␣
↪(thats what we've done in this tutorial)
# - Fit a simplier model (say a sersic or exponential instead of nonparametric)␣
↪first, then use that to initialize the complex model
# - Mix and match optimizers, if one gets stuck another may be better suited␣
↪for that area of parameter space
```

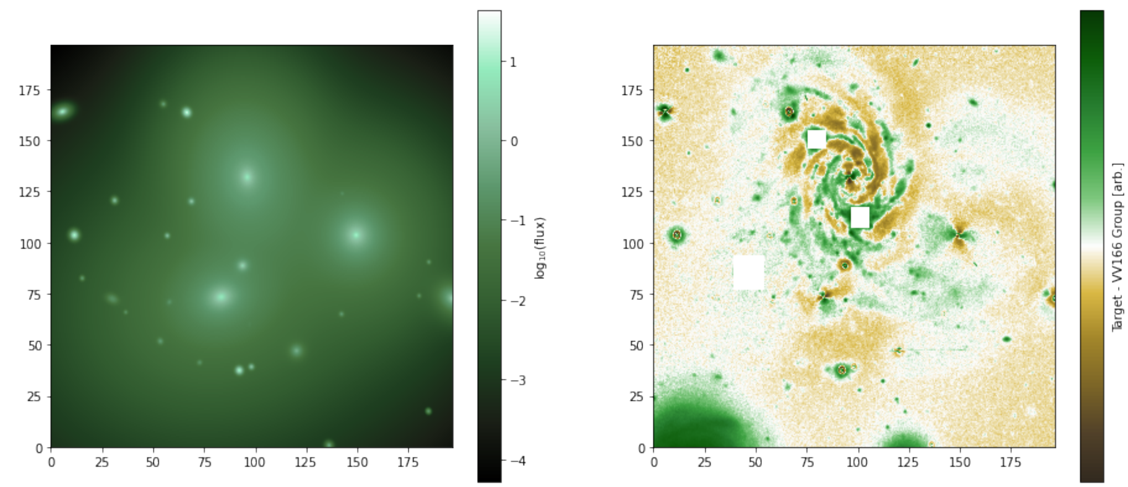
```
--------iter-------
sky
NGC0070
NGC0071
NGC0068
litte 1
litte 2
litte 3
minor object 02
minor object 03
minor object 04
minor object 05
minor object 06
minor object 08
minor object 10
minor object 11
minor object 12
minor object 13
minor object 14
minor object 15
minor object 16
minor object 17
```

```
minor object 19
minor object 21
minor object 22
minor object 25
minor object 26
minor object 27
minor object 30
minor object 31
Loss: 2.015694840723521
--------iter-------
sky
NGC0070
NGC0071
NGC0068
litte 1
litte 2
litte 3
minor object 02
minor object 03
minor object 04
minor object 05
minor object 06
minor object 08
minor object 10
minor object 11
minor object 12
minor object 13
minor object 14
minor object 15
minor object 16
minor object 17
minor object 19
minor object 21
minor object 22
minor object 25
minor object 26
minor object 27
minor object 30
minor object 31
Loss: 2.014898906678377
--------iter-------
sky
NGC0070
NGC0071
NGC0068
litte 1
litte 2
litte 3
```

```
minor object 02
minor object 03
minor object 04
minor object 05
minor object 06
minor object 08
minor object 10
minor object 11
minor object 12
minor object 13
minor object 14
minor object 15
minor object 16
minor object 17
minor object 19
minor object 21
minor object 22
minor object 25
minor object 26
minor object 27
minor object 30
minor object 31
Loss: 2.014853619391809
--------iter-------
sky
NGC0070
NGC0071
NGC0068
litte 1
litte 2
litte 3
minor object 02
minor object 03
minor object 04
minor object 05
minor object 06
minor object 08
minor object 10
minor object 11
minor object 12
minor object 13
minor object 14
minor object 15
minor object 16
minor object 17
minor object 19
minor object 21
minor object 22
```

```
minor object 25
minor object 26
minor object 27
minor object 30
minor object 31
Loss: 2.0148389598977587
success
```

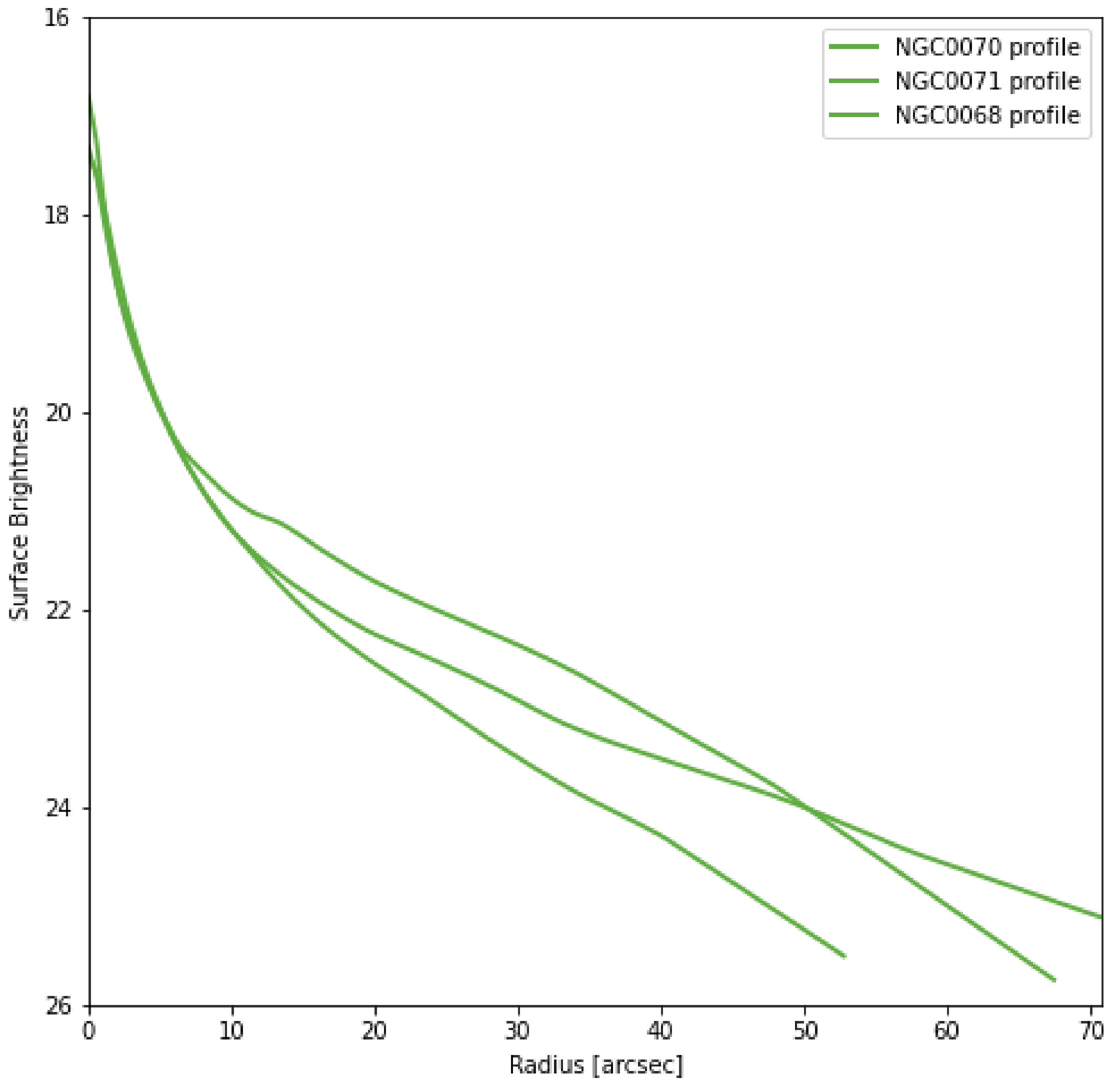
[18]:
```python
# Indeed the fit converges successfully! These tricks are really useful for complex fits.

# Now we can see what the fitting has produced
fig10, ax10 = plt.subplots(1,2,figsize = (16,7))
ap.plots.model_image(fig10, ax10[0], VV166Group)
ap.plots.residual_image(fig10, ax10[1], VV166Group)
plt.show()
```



Now that's starting to look like a complete model, and the Chi^2/ndf is much lower! And all for very little effort considering the level of detail. Looking at the residuals there is a clear improvement from the other attempts, that said there is a lot of structure in the residuals around the small objects, suggesting that a sersic alone is not the best model for these galaxies. That's not too surprising, at the very least we should apply PSF convolution to the models to get the proper blurring. PSF convolution is very slow though, so it would be best to do on a GPU, which you can try out if you have access to one! Simply set psf_mode = "full" and run fit again. For now though, we'll forgo the PSF convolution in the interest of time.

[19]:
```python
# and we can also take a look at the three main object profiles

fig8, ax8 = plt.subplots(figsize = (8,8))
# let's just plot the 3 main object profiles
for M in VV166Group.model_list[1:4]:
```

```
    ap.plots.galaxy_light_profile(fig8, ax8, M)
ax8.legend()
ax8.set_ylim([26,16])
plt.show()
```



[ ]: