JointModels

April 1, 2023

1 Joint Modelling

In this tutorial you will learn how to set up a joint modelling fit which encoporates the data from multiple images. These use `Group_Model` objects just like in the `GroupModels.ipynb` tutorial, the main difference being how the `Target_Image` object is constructed and that more care must be taken when assigning targets to models.

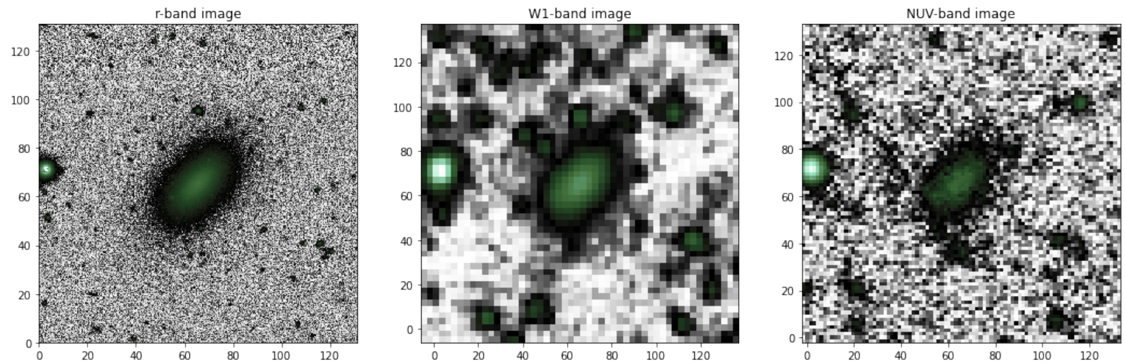
It is, of course, more work to set up a fit across multiple target images. However, the tradeoff can be well worth it. Perhaps there is space-based data with high resolution, but groundbased data has better S/N. Or perhaps each band individually does not have enough signal for a confident fit, but all three together just might. Perhaps colour information is of paramount importance for a science goal, one would hope that both bands could be treated on equal footing but in a consistent way when extracting profile information. There are a number of reasons why one might wish to try and fit a multi image picture of a galaxy simultaneously.

When fitting multiple bands one often resorts to forced photometry, somtimes also blurring each image to the same approximate PSF. With AutoProf this is entirely unecessary as one can fit each image in its native PSF simultaneously. The final fits are more meaningful and can encorporate all of the available structure information.

[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 we need some data to work with, let's use LEDA 41136 as our example␣
↪galaxy

# Our first image is from the DESI Legacy-Survey r-band. This image has a␣
↪pixelscale of 0.262 arcsec/pixel and is 500 pixels across
target_r = ap.image.Target_Image(
    data = np.array(fits.open("https://www.legacysurvey.org/viewer/fits-cutout?
↪ra=187.3119&dec=12.9783&size=500&layer=ls-dr9&pixscale=0.262&bands=r")[0].
↪data, dtype = np.float64),
    pixelscale = 0.262,
    zeropoint = 22.5,
```

```
    variance = np.ones((500,500))*0.008**2, # note that the variance is important to ensure all images are compared with proper statistical weight. Here we just use the IQR^2 of the pixel values as the variance, for science data one would use a more accurate variance value
    psf = ap.utils.initialize.gaussian_psf(1.12/2.355, 51, 0.262) # we construct a basic gaussian psf for each image by giving the simga (arcsec), image width (pixels), and pixelscale (arcsec/pixel)
)

# The second image is a unWISE W1 band image. This image has a pixelscale of 2.75 arcsec/pixel and is 52 pixels across
target_W1 = ap.image.Target_Image(
    data = np.array(fits.open("https://www.legacysurvey.org/viewer/fits-cutout?ra=187.3119&dec=12.9783&size=52&layer=unwise-neo7&pixscale=2.75&bands=1")[0].data, dtype = np.float64),
    pixelscale = 2.75,
    zeropoint = 25.199,
    variance = np.ones((52,52))*4.9**2,
    psf = ap.utils.initialize.gaussian_psf(6.1/2.355, 21, 2.75),
    origin = (np.array([500,500]))*0.262/2 - (np.array([52,52]))*2.75/2, # here we ensure that the images line up by slightly adjusting the origin
)

# The third image is a GALEX NUV band image. This image has a pixelscale of 1.5 arcsec/pixel and is 90 pixels across
target_NUV = ap.image.Target_Image(
    data = np.array(fits.open("https://www.legacysurvey.org/viewer/fits-cutout?ra=187.3119&dec=12.9783&size=90&layer=galex&pixscale=1.5&bands=n")[0].data, dtype = np.float64),
    pixelscale = 1.5,
    zeropoint = 20.08,
    variance = np.ones((90,90))*0.0007**2,
    psf = ap.utils.initialize.gaussian_psf(5.4/2.355, 21, 1.5),
    origin = (np.array([500,500]))*0.262/2 - (np.array([90,90]))*1.5/2,
)

fig1, ax1 = plt.subplots(1, 3, figsize = (18,6))
ap.plots.target_image(fig1, ax1[0], target_r)
ax1[0].set_title("r-band image")
ap.plots.target_image(fig1, ax1[1], target_W1)
ax1[1].set_title("W1-band image")
ap.plots.target_image(fig1, ax1[2], target_NUV)
ax1[2].set_title("NUV-band image")
plt.show()
```



[3]:
```python
# The joint model will need a target to try and fit, but now that we have multiple images the "target" is
# a Target_Image_List object which points to all three.
target_full = ap.image.Target_Image_List((target_r, target_W1, target_NUV))
# It doesn't really need any other information since everything is already available in the individual targets
```

[4]:
```python
# To make things easy to start, lets just fit a sersic model to all three. In principle one can use arbitrary
# group models designed for each band individually, but that would be unecessarily complex for a tutorial

model_r = ap.models.AutoProf_Model(
    name = "rband model",
    model_type = "sersic galaxy model",
    target = target_r,
    psf_mode = "full",
)
model_W1 = ap.models.AutoProf_Model(
    name = "W1band model",
    model_type = "sersic galaxy model",
    target = target_W1,
    psf_mode = "full",
)
model_NUV = ap.models.AutoProf_Model(
    name = "NUVband model",
    model_type = "sersic galaxy model",
    target = target_NUV,
    psf_mode = "full",
)

# At this point we would just be fitting three separate models at the same time, not very interesting. Next
```

```python
# we add constraints so that some parameters are shared between all the models.␣
↪It makes sense to fix
# structure parameters while letting brightness parameters vary between bands␣
↪so that's what we do here.
model_W1.add_equality_constraint(model_r, ["center", "q", "PA", "n", "Re"])
model_NUV.add_equality_constraint(model_r, ["center", "q", "PA", "n", "Re"])
# Now every model will have a unique Ie, but every other parameter is shared␣
↪for all three
```

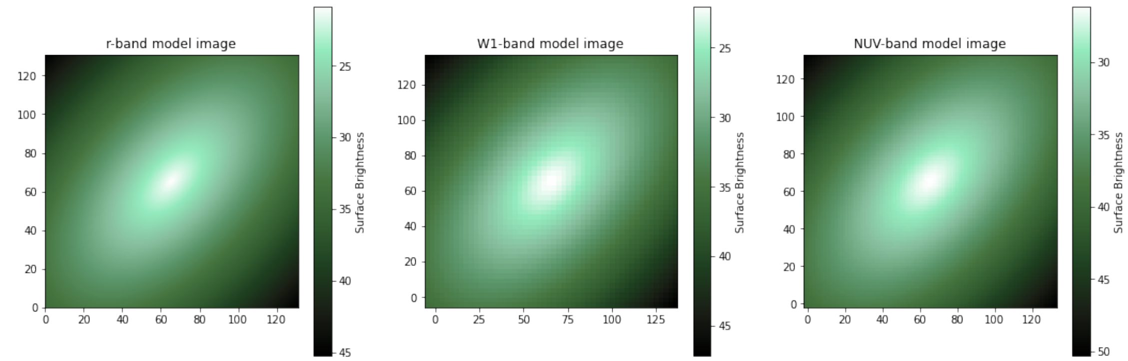
[5]:
```python
# We can now make the joint model object

model_full = ap.models.AutoProf_Model(
    name = "LEDA 41136",
    model_type = "group model",
    model_list = [model_r, model_W1, model_NUV],
    target = target_full,
)

model_full.initialize()
```

```
/home/connor/Programming/AutoProf-2/autoprof/utils/parametric_profiles.py:38:
RuntimeWarning: overflow encountered in exp
  return Ie * np.exp(-bn * ((R / Re) ** (1 / n) - 1))
```

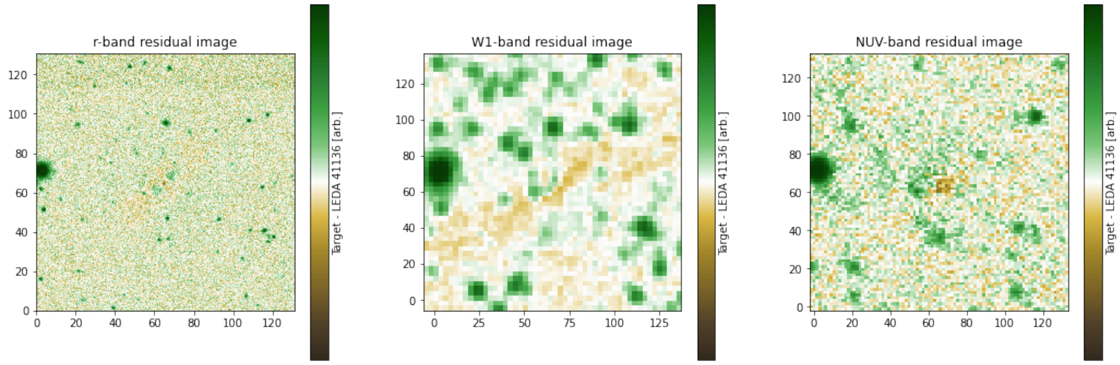
[6]:
```python
result = ap.fit.LM(model_full, verbose = 1).fit()
print(result.message)
```

```
L: 1.0
---------init---------
LM loss: 93.27173207684267
L: 1.0
---------iter---------
LM loss: 93.26164285665872
accept
L: 0.1111111111111111
---------iter---------
LM loss: 93.2485056173143
accept
L: 0.012345679012345678
---------iter---------
LM loss: 93.2446531273942
accept
L: 0.0013717421124828531
---------iter---------
LM loss: 93.24390336248476
accept
success
```

[7]:
```
# here we plot the results of the fitting, notice that each band has a␣
↪different PSF and pixelscale. Also, notice
# that the colour bars represent significantly different ranges since each␣
↪model was allowed to fit its own Ie.
# meanwhile the center, PA, q, and Re is the same for every model.
fig1, ax1 = plt.subplots(1, 3, figsize = (18,6))
ap.plots.model_image(fig1, ax1, model_full)
ax1[0].set_title("r-band model image")
ax1[1].set_title("W1-band model image")
ax1[2].set_title("NUV-band model image")
plt.show()
```



[8]:
```
# We can also plot the residual images. As can be seen, the galaxy is fit in␣
↪all three bands simultaneously
# with the majority of the light removed in all bands. A residual can be seen␣
↪in the r band. This is likely
# due to there being more structure in the r-band than just a sersic. The W1␣
↪and NUV bands look excellent though
fig1, ax1 = plt.subplots(1, 3, figsize = (18,6))
ap.plots.residual_image(fig1, ax1, model_full)
ax1[0].set_title("r-band residual image")
ax1[1].set_title("W1-band residual image")
ax1[2].set_title("NUV-band residual image")
plt.show()
```



1.1 Joint models with multiple models

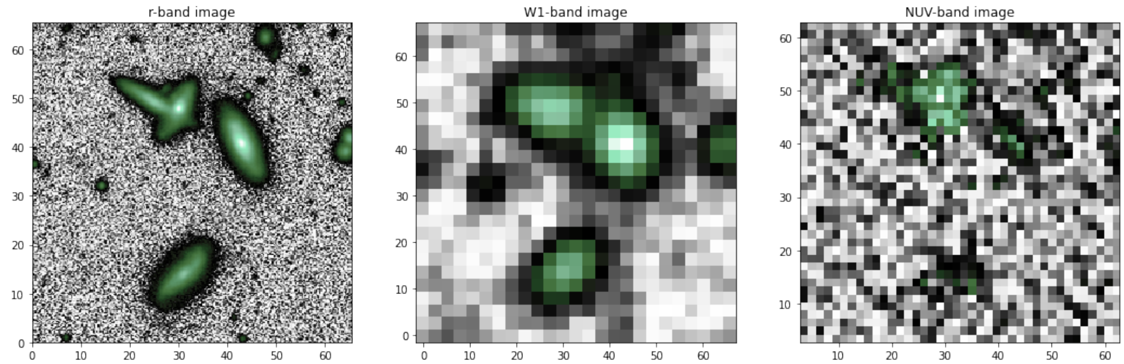
If you want to analyze more than a single astronomical object, you will need to combine many models for each image in a reasonable structure. There are a number of ways to do this that will work, though may not be as scalable. For small images, just about any arrangement is fine when using the LM optimizer. But as images and number of models scales very large, it may be neccessary to sub divide the problem to save memory. To do this you should arrange your models in a hierarchy so that AutoProf has some information about the structure of your problem. There are two ways to do this. First, you can create a group of models where each sub-model is a group which holds all the objects for one image. Second, you can create a group of models where each sub-model is a group which holds all the representations of a single astronomical object across each image. The second method is preferred. See the diagram below to help clarify what this means.

JointGroupModels

Here we will see an example of a multiband fit of an image which has multiple astronomical objects.

[9]:
```
# First we need some data to work with, let's use another LEDA object, this time a group of galaxies: LEDA 389779, 389797, 389681

RA = 320.5003
DEC = -57.4585
# Our first image is from the DESI Legacy-Survey r-band. This image has a pixelscale of 0.262 arcsec/pixel
rsize = 250
target_r = ap.image.Target_Image(
    data = np.array(fits.open(f"https://www.legacysurvey.org/viewer/fits-cutout?ra={RA}&dec={DEC}&size={rsize}&layer=ls-dr9&pixscale=0.262&bands=r")[0].data, dtype = np.float64),
    pixelscale = 0.262,
    zeropoint = 22.5,
    variance = np.ones((rsize,rsize))*0.008**2, # note that the variance is important to ensure all images are compared with proper statistical weight. Here we just use the IQR^2 of the pixel values as the variance, for science data one would use a more accurate variance value
```

```
    psf = ap.utils.initialize.gaussian_psf(1.12/2.355, 51, 0.262) # we construct a basic gaussian psf for each image by giving the simga (arcsec), image width (pixels), and pixelscale (arcsec/pixel)
)

# The second image is a unWISE W1 band image. This image has a pixelscale of 2.75 arcsec/pixel
wsize = 25
target_W1 = ap.image.Target_Image(
    data = np.array(fits.open(f"https://www.legacysurvey.org/viewer/fits-cutout?ra={RA}&dec={DEC}&size={wsize}&layer=unwise-neo7&pixscale=2.75&bands=1")[0].data, dtype = np.float64),
    pixelscale = 2.75,
    zeropoint = 25.199,
    variance = np.ones((wsize,wsize))*4.9**2,
    psf = ap.utils.initialize.gaussian_psf(6.1/2.355, 21, 2.75),
    origin = (np.array([rsize,rsize]))*0.262/2 - (np.array([wsize,wsize]))*2.75/2, # here we ensure that the images line up by slightly adjusting the origin
)

# The third image is a GALEX NUV band image. This image has a pixelscale of 1.5 arcsec/pixel
gsize = 40
target_NUV = ap.image.Target_Image(
    data = np.array(fits.open(f"https://www.legacysurvey.org/viewer/fits-cutout?ra={RA}&dec={DEC}&size={gsize}&layer=galex&pixscale=1.5&bands=n")[0].data, dtype = np.float64),
    pixelscale = 1.5,
    zeropoint = 20.08,
    variance = np.ones((gsize,gsize))*0.0007**2,
    psf = ap.utils.initialize.gaussian_psf(5.4/2.355, 21, 1.5),
    origin = (np.array([rsize,rsize]))*0.262/2 - (np.array([gsize,gsize]))*1.5/2,
)
target_full = ap.image.Target_Image_List((target_r, target_W1, target_NUV))

fig1, ax1 = plt.subplots(1, 3, figsize = (18,6))
ap.plots.target_image(fig1, ax1, target_full)
ax1[0].set_title("r-band image")
ax1[1].set_title("W1-band image")
ax1[2].set_title("NUV-band image")
plt.show()
```

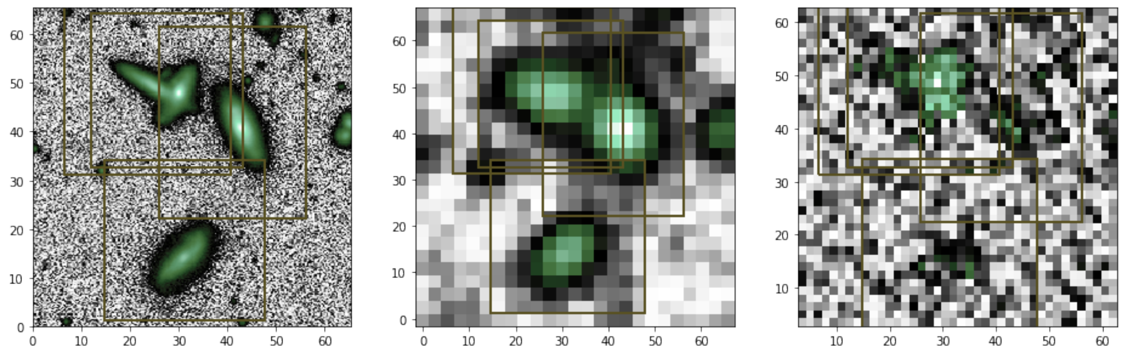


There is barely any signal in the GALEX data and it would be entirely impossible to analyze on its own. With simultaneous multiband fitting it is a breeze to get relatively robust results!

Next we need to construct models for each galaxy. This is understandably more complex than in the single band case, since now we have three times the amout of data to keep track of. Recall that we will create a number of joint models to represent each astronomical object, then put them all together in a larger group model.

[10]:
```
# Here we enter the window parameters by hand, in general one would use a␣
↪segmentation map or some other automated proceedure to pick out the area for␣
↪many objects
windows = [
    {"r":[[72,152],[140,234]], "W1": [[5,16],[13,24]], "NUV": [[8,27],[20,39]]},
    {"r":[[43,155],[138,237]], "W1": [[3,15],[12,25]], "NUV": [[4,22],[19,39]]},
    {"r":[[115,210],[100,228]], "W1": [[10,21],[10,23]], "NUV":␣
↪[[17,35],[13,38]]},
    {"r":[[69,170],[10,115]], "W1": [[7,17],[1,13]], "NUV": [[8,30],[1,18]]},
]

model_list = []

for i, window in enumerate(windows):
    # create the submodels for this object
    sub_list = []
    sub_list.append(
        ap.models.AutoProf_Model(
            name = f"rband model {i}",
            model_type = "spline galaxy model", # we use spline models for the␣
↪r-band since it is well resolved
            target = target_r,
            window = window["r"],
            psf_mode = "full",
        )
    )
```

```
    sub_list.append(
        ap.models.AutoProf_Model(
            name = f"W1band model {i}",
            model_type = "sersic galaxy model", # we use sersic models for W1␣
↪and NUV since there isn't much visible detail, a simple model is sufficient
            target = target_W1,
            window = window["W1"],
            psf_mode = "full",
        )
    )
    sub_list.append(
        ap.models.AutoProf_Model(
            name = f"NUVband model {i}",
            model_type = "sersic galaxy model",
            target = target_NUV,
            window = window["NUV"],
            psf_mode = "full",
        )
    )
    # ensure equality constraints
    sub_list[1].add_equality_constraint(sub_list[0], ["center", "q", "PA"])
    sub_list[2].add_equality_constraint(sub_list[0], ["center", "q", "PA"])
    # Make the multiband model for this object
    model_list.append(
        ap.models.AutoProf_Model(
            name = f"model {i}",
            model_type = "group model",
            target = target_full,
            model_list = sub_list,
        )
    )
# Make the full model for this system of objects
MODEL = ap.models.AutoProf_Model(
    name = f"full model",
    model_type = "group model",
    target = target_full,
    model_list = model_list,
)
fig, ax = plt.subplots(1,3, figsize = (16,7))
ap.plots.target_image(fig, ax, MODEL.target)
ap.plots.model_window(fig, ax, MODEL)
ax1[0].set_title("r-band image")
ax1[1].set_title("W1-band image")
ax1[2].set_title("NUV-band image")
plt.show()
```



[11]:
```
MODEL.initialize()
result = ap.fit.LM(MODEL, verbose = 1, epsilon4 = 0.05).fit()
print(result.message)
```

```
/home/connor/Programming/AutoProf-2/autoprof/utils/parametric_profiles.py:38:
RuntimeWarning: overflow encountered in exp
  return Ie * np.exp(-bn * ((R / Re) ** (1 / n) - 1))
/home/connor/Programming/AutoProf-2/autoprof/models/_shared_methods.py:129:
RuntimeWarning: divide by zero encountered in log10
  residual = (f - np.log10(prof_func(r, *x))) ** 2
/home/connor/Programming/AutoProf-2/autoprof/utils/parametric_profiles.py:38:
RuntimeWarning: overflow encountered in exp
  return Ie * np.exp(-bn * ((R / Re) ** (1 / n) - 1))
/home/connor/Programming/AutoProf-2/autoprof/utils/parametric_profiles.py:38:
RuntimeWarning: overflow encountered in exp
  return Ie * np.exp(-bn * ((R / Re) ** (1 / n) - 1))
/home/connor/Programming/AutoProf-2/autoprof/models/_shared_methods.py:129:
RuntimeWarning: divide by zero encountered in log10
  residual = (f - np.log10(prof_func(r, *x))) ** 2
/home/connor/Programming/AutoProf-2/autoprof/utils/parametric_profiles.py:38:
RuntimeWarning: overflow encountered in exp
  return Ie * np.exp(-bn * ((R / Re) ** (1 / n) - 1))
/home/connor/Programming/AutoProf-2/autoprof/models/_shared_methods.py:129:
RuntimeWarning: divide by zero encountered in log10
  residual = (f - np.log10(prof_func(r, *x))) ** 2
```

```
L: 1.0
---------init---------
LM loss: 6.541543282085514
L: 1.0
---------iter---------
LM loss: 929173.4560658981
reject
L: 11.0
---------iter---------
```

```
LM loss: 5.561268870814513
accept
L: 1.2222222222222223
---------iter---------
LM loss: 22216.910836756786
reject
L: 13.444444444444446
---------iter---------
LM loss: 5.11051354197551
accept
L: 1.4938271604938274
---------iter---------
LM loss: 5809.781146532198
reject
L: 16.4320987654321
---------iter---------
LM loss: 4.796279529435459
accept
L: 1.825788751714678
---------iter---------
LM loss: 20.858507826094705
reject
L: 20.08367626886146
---------iter---------
LM loss: 4.592759781235849
accept
L: 2.231519585429051
---------iter---------
LM loss: 3.4563856003584226
accept
L: 0.2479466206032279
---------iter---------
LM loss: 106314.43985164943
reject
L: 2.727412826635507
---------iter---------
LM loss: 3.034928257596934
accept
L: 0.3030458696261674
---------iter---------
LM loss: 107736.68280992967
reject
L: 3.3335045658878415
---------iter---------
LM loss: 2.78594381907817
accept
L: 0.3703893962097602
---------iter---------
```

```
LM loss: 107205.81882540115
reject
L: 4.074283358307362
---------iter---------
LM loss: 2.647998610196552
accept
L: 0.4526981509230402
---------iter---------
LM loss: 105270.3390931869
reject
L: 4.979679660153442
---------iter---------
LM loss: 2.55571737278968
accept
L: 0.5532977400170491
---------iter---------
LM loss: 99557.79501528101
reject
L: 6.08627514018754
---------iter---------
LM loss: 2.493246503746012
accept
L: 0.6762527933541711
---------iter---------
LM loss: 80858.2138731703
reject
L: 7.438780726895882
---------iter---------
LM loss: 2.4470596747587945
accept
L: 0.8265311918773202
---------iter---------
LM loss: 4557.404395169475
reject
L: 9.091843110650522
---------iter---------
LM loss: 2.3891667298629407
accept
L: 1.0102047900722804
---------iter---------
LM loss: 2.114320644556947
accept
L: 0.11224497667469782
---------iter---------
LM loss: 1.790503665742936
accept
L: 0.012471664074966424
---------iter---------
```

```
LM loss: 7612251358486.516
reject
L: 0.13718830482463065
---------iter---------
LM loss: 1.6406943597240047
accept
L: 0.015243144980514517
---------iter---------
LM loss: 18796.010735691067
reject
L: 0.1676745947856597
---------iter---------
LM loss: 1.8050216272943824
reject
L: 1.8444205426422566
---------iter---------
LM loss: 1.5833766839696155
accept
L: 0.20493561584913964
---------iter---------
LM loss: 1.4417531685268035
accept
L: 0.022770623983237738
---------iter---------
LM loss: 1.3188924890888465
accept
L: 0.0025300693314708597
---------iter---------
LM loss: 4.959610657768491e+25
reject
L: 0.027830762646179456
---------iter---------
LM loss: 451930857.07415116
reject
L: 0.30613838910797403
---------iter---------
LM loss: 1.2892548135274786
accept
L: 0.03401537656755267
---------iter---------
LM loss: 1.5304662594676488
reject
L: 0.3741691422430794
---------iter---------
LM loss: 1.2796812016459644
accept
L: 0.041574349138119936
---------iter---------
```

```
LM loss: 1.278784168189147
accept
L: 0.0046193721264577705
---------iter---------
LM loss: inf
nan loss
L: 0.05081309339103548
---------iter---------
LM loss: 1.2661016030131114
accept
L: 0.005645899265670609
---------iter---------
LM loss: inf
nan loss
L: 0.0621048919223767
---------iter---------
LM loss: 1.254085614449319
accept
L: 0.006900543546930744
---------iter---------
LM loss: inf
nan loss
L: 0.07590597901623819
---------iter---------
LM loss: 1.2211032562067444e+116
reject
L: 0.8349657691786201
---------iter---------
LM loss: 1.2525217935368895
accept
L: 0.09277397435318001
---------iter---------
LM loss: 2.3666943160476394e+78
reject
L: 1.0205137178849801
---------iter---------
LM loss: 1.2514060099344138
accept
L: 0.11339041309833113
---------iter---------
LM loss: 1.1345503789493347
accept
L: 0.01259893478870346
---------iter---------
LM loss: 1.1311345263734278
accept
L: 0.0013998816431892733
---------iter---------
```

```
LM loss: 1.0748776625077283
accept
L: 0.00015554240479880813
---------iter---------
LM loss: 1.5888070888347874
reject
L: 0.0017109664527868895
---------iter---------
LM loss: 1.0710002615611807
accept
L: 0.00019010738364298773
---------iter---------
LM loss: 14.103016799028001
reject
L: 0.002091181220072865
---------iter---------
LM loss: 6.506854064079468
reject
L: 0.023002993420801515
---------iter---------
LM loss: 1.340899275822673
reject
L: 0.2530329276288167
---------iter---------
LM loss: 1.026927743118788
accept
L: 0.02811476973653519
---------iter---------
LM loss: 1.0216115319040182
accept
L: 0.0031238633040594653
---------iter---------
LM loss: 1.0215140764017825
accept
L: 0.0003470959226732739
---------iter---------
LM loss: 1.061855983323741
reject
L: 0.0038180551494060126
---------iter---------
LM loss: 1.0194319814782251
accept
L: 0.0004242283499340014
---------iter---------
LM loss: 1.065012024720344
reject
L: 0.004666511849274016
---------iter---------
```

```
LM loss: 1.0192330582776385
accept
L: 0.0005185013165860018
---------iter---------
LM loss: 1.0514550833769116
reject
L: 0.00570351448244602
---------iter---------
LM loss: 1.0189074357124313
accept
L: 0.000633723831382891
---------iter---------
LM loss: 1.0472541250316265
reject
L: 0.006970962145211802
---------iter---------
LM loss: 1.0186523949693471
accept
L: 0.0007745513494679779
---------iter---------
LM loss: 1.0565113354772553
reject
L: 0.008520064844147758
---------iter---------
LM loss: 1.0131855518282782
accept
L: 0.0009466738715719731
---------iter---------
LM loss: 1.0313377399177377
reject
L: 0.010413412587291703
---------iter---------
LM loss: 0.9991871591714799
accept
L: 0.0011570458430324114
---------iter---------
LM loss: 1.0029576989427087
reject
L: 0.012727504273356526
---------iter---------
LM loss: 0.9936540131224089
accept
L: 0.0014141671414840584
---------iter---------
LM loss: 0.9988786160052999
reject
L: 0.015555838556324643
---------iter---------
```

```
LM loss: 0.9929331898885796
accept
L: 0.0017284265062582937
---------iter---------
LM loss: 0.9928608149250129
reject
L: 0.01901269156884123
---------iter---------
LM loss: 0.9918549892926821
accept
L: 0.0021125212854268033
---------iter---------
LM loss: 0.9922222655944917
reject
L: 0.023237734139694835
---------iter---------
LM loss: 0.9917527912594251
accept
L: 0.002581970459966093
---------iter---------
LM loss: 0.9918244913402269
reject
L: 0.02840167505962702
---------iter---------
LM loss: 0.9918241659870107
reject
L: 0.3124184256558972
---------iter---------
LM loss: 0.9919276868854531
reject
L: 3.4366026822148696
---------iter---------
LM loss: 0.99182053107269
reject
L: 37.802629504363566
---------iter---------
LM loss: 0.9917582461829034
reject
L: 415.82892454799924
---------iter---------
LM loss: 0.9917532663496098
reject
L: 4574.118170027991
---------iter---------
LM loss: 0.9917528345073173
reject
L: 50315.29987030791
---------iter---------
```

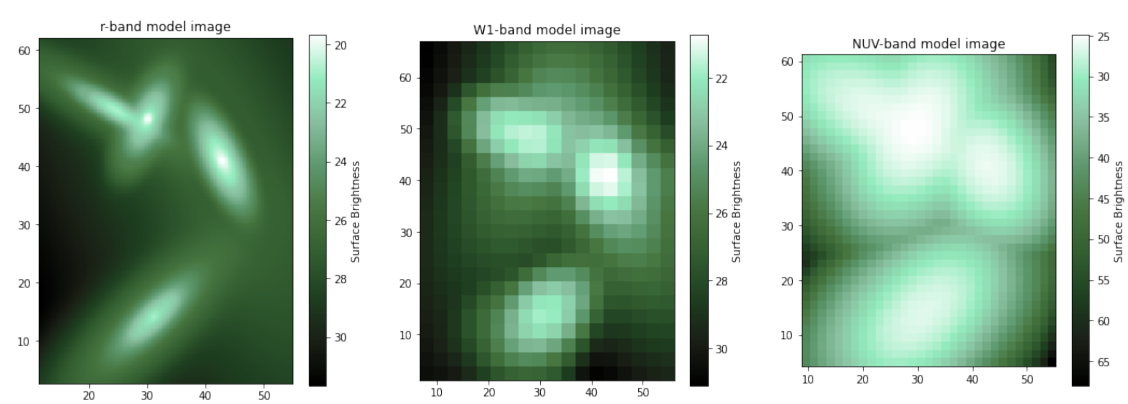
```
LM loss: 0.9917527953593904
reject
L: 553468.298573387
---------iter---------
LM loss: 0.9917527916765799
reject
L: 6088151.284307256
---------iter---------
LM loss: 0.9917527913025607
reject
L: 66969664.12737982
---------iter---------
LM loss: 0.9917527912636964
reject
L: 736666305.401178
---------iter---------
LM loss: 0.9917527912595815
reject
accept grad
L: 1.0
---------iter---------
LM loss: 0.9918961669248014
reject
L: 11.0
---------iter---------
LM loss: 0.9917696222080878
reject
L: 121.0
---------iter---------
LM loss: 0.9917501665076645
reject
L: 1331.0
---------iter---------
LM loss: 0.9917485936697124
reject
L: 14641.0
---------iter---------
LM loss: 0.9917484527003687
reject
L: 161051.0
---------iter---------
LM loss: 0.9917484397962123
reject
L: 1771561.0
---------iter---------
LM loss: 0.991748438534271
reject
L: 19487171.0
```

```
---------iter---------
LM loss: 0.9917484384058929
reject
L: 214358881.0
---------iter---------
LM loss: 0.9917484383928891
reject
accept bad grad
L: 1.0
---------iter---------
LM loss: 0.9918967742470524
reject
L: 11.0
---------iter---------
LM loss: 0.9917702181814877
reject
L: 121.0
---------iter---------
LM loss: 0.9917507556496267
reject
L: 1331.0
---------iter---------
LM loss: 0.9917491821053429
reject
L: 14641.0
---------iter---------
LM loss: 0.9917490410708221
reject
L: 161051.0
---------iter---------
LM loss: 0.9917490281597068
reject
L: 1771561.0
---------iter---------
LM loss: 0.9917490268977125
reject
L: 19487171.0
---------iter---------
LM loss: 0.9917490267697451
reject
L: 214358881.0
---------iter---------
LM loss: 0.9917490267562862
reject
accept bad grad
L: 1.0
---------iter---------
LM loss: 0.9918968356687887
```

```
reject
L: 11.0
---------iter---------
LM loss: 0.9917702784669706
reject
L: 121.0
---------iter---------
LM loss: 0.9917508152536189
reject
L: 1331.0
---------iter---------
LM loss: 0.9917492416380024
reject
L: 14641.0
---------iter---------
LM loss: 0.9917491005963948
reject
L: 161051.0
---------iter---------
LM loss: 0.9917490876865721
reject
L: 1771561.0
---------iter---------
LM loss: 0.9917490864230621
reject
L: 19487171.0
---------iter---------
LM loss: 0.9917490862953285
reject
L: 214358881.0
---------iter---------
LM loss: 0.9917490862816314
reject
accept bad grad
L: 1.0
---------iter---------
LM loss: 0.9918968972149362
reject
L: 11.0
---------iter---------
LM loss: 0.9917703388795663
reject
L: 121.0
---------iter---------
LM loss: 0.9917508749815708
reject
L: 1331.0
---------iter---------
```

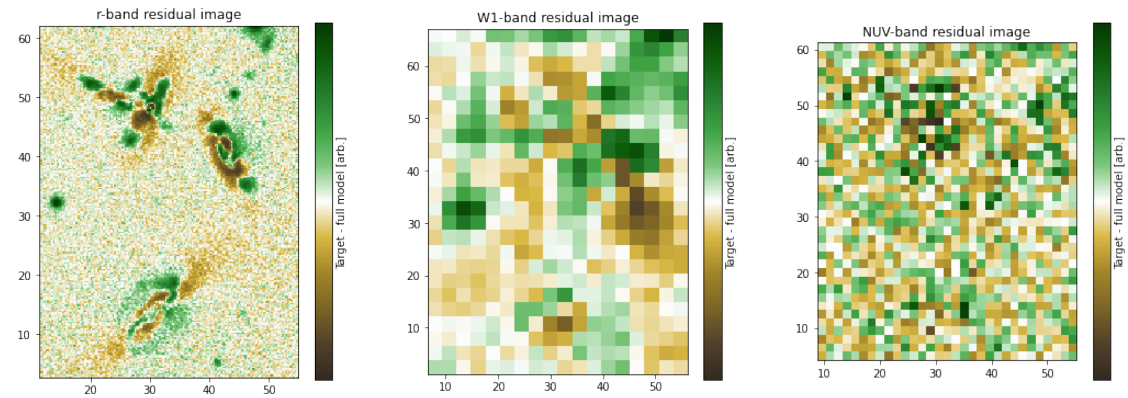
```
LM loss: 0.9917493012957685
reject
L: 14641.0
---------iter---------
LM loss: 0.9917491602482741
reject
L: 161051.0
---------iter---------
LM loss: 0.9917491473359836
reject
L: 1771561.0
---------iter---------
LM loss: 0.9917491460738868
reject
L: 19487171.0
---------iter---------
LM loss: 0.9917491459459113
reject
L: 214358881.0
---------iter---------
LM loss: 0.9917491459324523
reject
accept bad grad
L: 1.0
---------iter---------
LM loss: 0.9918969588872429
reject
success by immobility, unable to find improvement either converged or bad area
of parameter space.
```

[12]:
```python
fig1, ax1 = plt.subplots(1, 3, figsize = (18,6))
ap.plots.model_image(fig1, ax1, MODEL)
ax1[0].set_title("r-band model image")
ax1[1].set_title("W1-band model image")
ax1[2].set_title("NUV-band model image")
plt.show()
```



The models look excellent! The power of multiband fitting lets us know that we have extracted all the available information here, no forced photometry required!

[13]:
```
fig, ax = plt.subplots(1, 3, figsize = (18,6))
ap.plots.residual_image(fig, ax, MODEL)
ax[0].set_title("r-band residual image")
ax[1].set_title("W1-band residual image")
ax[2].set_title("NUV-band residual image")
plt.show()
```



The residuals look acceptable, but clearly there is more structure to be found in these galaxies, this is especially apparent in the r-band data. At least for the lower galaxy, we can see in the observed image that there are spiral arms, those can easily cause large scale residual patterns.

1.1.1 Dithered images

Note that it is not necessary to use images from different bands. Using dithered images one can effectively achieve higher resolution. It is possible to simultaneously fit dithered images with AutoProf instead of postprocessing the two images together. This will of course be slower, but may be worthwhile for cases where extra care is needed.

1.1.2 Stacked images

Like dithered images, one may wish to combine the statistical power of multiple images but for some reason it is not clear how to add them. In this case one can simply have AutoProf fit the images simultaneously. Again this is slower than if the image could be combined, but should extract all the statistical power from the data.

1.1.3 Time series

Some objects change over time. For example they may get brighter and dimmer, or may have a transient feature appear. However, the structure of an object may remain constant. An example of this is a supernova and its host galaxy. The host galaxy likely doesn't change across images, but the supernova does. It is possible to fit a time series dataset with a shared galaxy model across multiple images, and a shared position for the supernova, but a variable brightness for the supernova over each image.

It is possible to get quite creative with joint models as they allow one to fix selective features of a model over a wide range of data. If you have a situation which may benefit from joint modelling but are having a hard time determining how to format everything, please do contact us!

[ ]:
```

```