AdvancedPSFModels

June 17, 2023

1 Advanced PSF modeling

Ideally we always have plenty of well separated bright, but not oversaturated, stars to use to construct a PSF model. These models are incredibly important for certain science objectives that rely on precise shape measurements and not just total light measures. Here we demonstrate some of the special capabilities AutoPhot has to handle challenging scenarios where a good PSF model is needed but there are only very faint stars, poorly placed stars, or even no stars to work with!

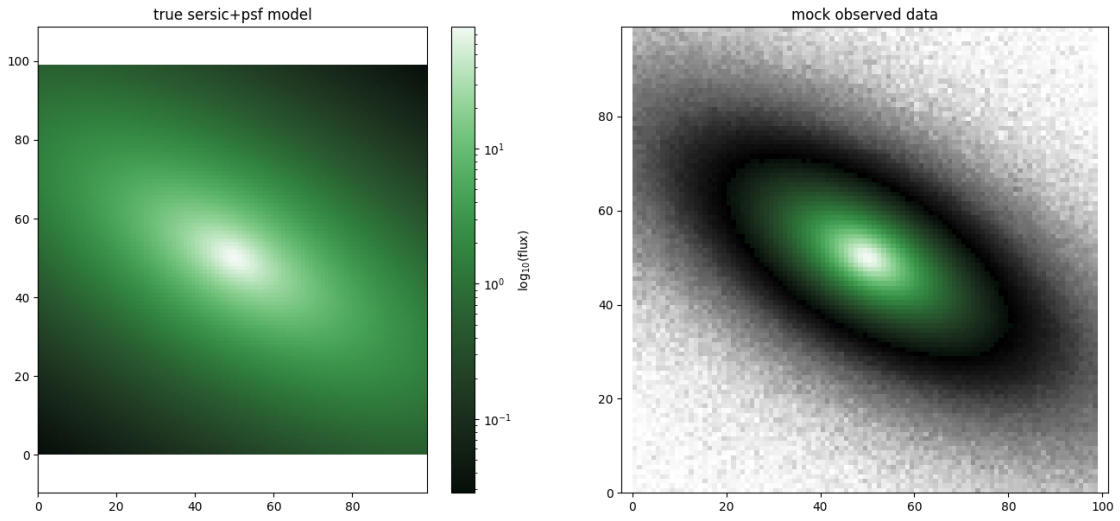
[1]:
```
import autophot as ap
import numpy as np
import torch
from astropy.io import fits
import matplotlib.pyplot as plt
from time import time
%matplotlib inline
```

2 PSF modeling without stars

Can it be done? Let's see!

[2]:
```
# Lets make some data that we need to fit

true_psf = ap.utils.initialize.moffat_psf(
    2., # n                                   !!!!! Take note, we want to get n =
 ↪2. !!!!!!
    3., # Rd                                  !!!!! Take note, we want to get Rd =
 ↪3.!!!!!!
    51, # pixels
    1.  # pixelscale
)

target = ap.image.Target_Image(
    data = torch.zeros(99,99),
    pixelscale = 1.,
    psf = true_psf,
)
```

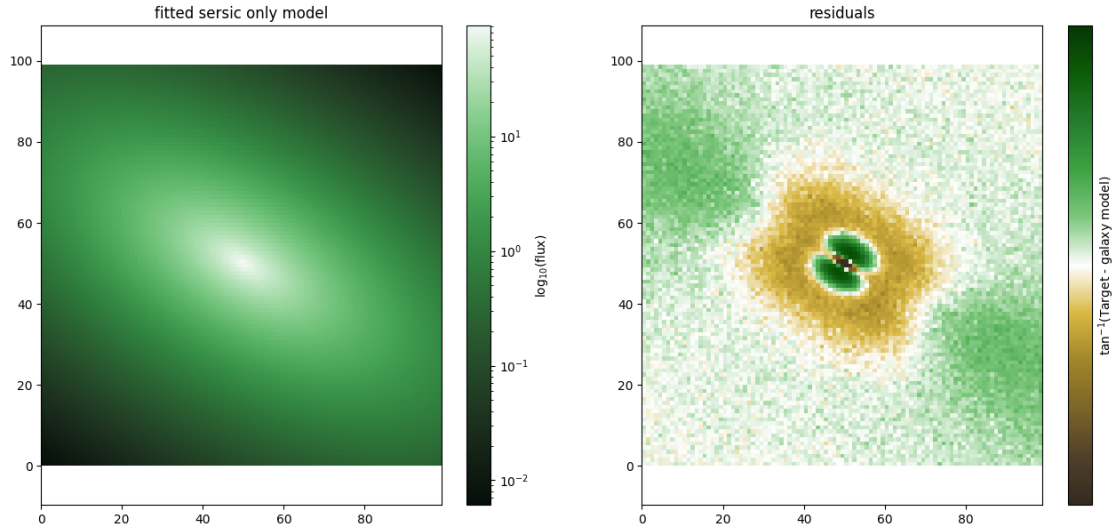
```python
true_model = ap.models.AutoPhot_Model(
    name = "true model",
    model_type = "sersic galaxy model",
    target = target,
    parameters = {
        "center": [50.,50.],
        "q": 0.4,
        "PA": np.pi/3,
        "n": 2,
        "Re": 25,
        "Ie": 1,
    },
    psf_mode = "full",
    #integrate_mode = "none",
)

# use the true model to make some data
sample = true_model()
torch.manual_seed(61803398)
target.data = sample.data + torch.normal(torch.zeros_like(sample.data), 0.1)
target.variance = 0.01*torch.ones_like(sample.data)

fig, ax = plt.subplots(1,2, figsize = (16,7))
ap.plots.model_image(fig, ax[0], true_model)
ap.plots.target_image(fig, ax[1], target)
ax[0].set_title("true sersic+psf model")
ax[1].set_title("mock observed data")
plt.show()
```



[3]:
```
# Now we will try and fit the data using just a plain sersic

# Here we set up a sersic model for the galaxy
plain_galaxy_model = ap.models.AutoPhot_Model(
    name = "galaxy model",
    model_type = "sersic galaxy model",
    target = target,
)

# Let AutoPhot determine its own intial parameters, so it has to start with␣
 ↪whatever it decides automatically,
# just like a real fit.
plain_galaxy_model.initialize()

result = ap.fit.LM(plain_galaxy_model, verbose = 1).fit()
print(result.message)
```

```
L: 1.0
---------init---------
LM loss: 242.11263073153592
L: 1.0
---------iter---------
LM loss: 43.52400611508354
accept
L: 0.1111111111111111
---------iter---------
LM loss: 23.80228736003797
accept
L: 0.012345679012345678
---------iter---------
LM loss: 23.207478106624958
accept
L: 0.0013717421124828531
---------iter---------
LM loss: 23.173956423895138
accept
L: 0.00015241579027587256
---------iter---------
LM loss: 23.17299574064802
accept
L: 1.6935087808430286e-05
---------iter---------
LM loss: 23.172981589530302
accept
success
```

[4]:
```
# The shape of the residuals here shows that there is still missing information;
 ↪ this is of course
# from the missing PSF convolution to blur the model. In fact, the shape of␣
 ↪those residuals is very
# commonly seen in real observed data (ground based) when it is fit without␣
 ↪accounting for PSF blurring.
fig, ax = plt.subplots(1,2, figsize = (16,7))
ap.plots.model_image(fig, ax[0], plain_galaxy_model)
ap.plots.residual_image(fig, ax[1], plain_galaxy_model)
ax[0].set_title("fitted sersic only model")
ax[1].set_title("residuals")
plt.show()
```



[5]:
```
# Now we will try and fit the data with a sersic model and a "live" psf

# Here we set up a sersic model for the galaxy
live_galaxy_model = ap.models.AutoPhot_Model(
    name = "galaxy model",
    model_type = "sersic galaxy model",
    target = target,
    psf_mode = "full",
    psf_subpixel_shift = False, # we turn this off for numerical stability␣
 ↪since the true position is exactly 50,50 which is between 4 pixels.
)

# Here we create a moffat model for the PSF. Note that this is just a regular␣
 ↪AutoPhot model that we have chosen
# to be a moffat, really any model can be used. To make it suitable as a PSF we␣
 ↪will need to apply some very
```

```
# specific settings. The window for the model is now just used to set the size␣
 ↪of the PSF (51 by 51 pixels).
# The "center" must be put at the center of the window (note that this is a bit␣
 ↪harder when pixelscale != 1).
# The "I0" value doesn't matter for a PSF (which will be normalized␣
 ↪internally), we pick I0 = 1 just for
# numerical stability. The "psf_upscale" parameter is set to 1 just to indicate␣
 ↪we are not doing any super
# resolution fitting this time.
live_psf_model = ap.models.AutoPhot_Model(
    name = "psf",
    model_type = "moffat star model",
    target = target,
    window = [[0,51],[0,51]],
    parameters = {
        "n": 1., # True value is 2.
        "Rd": 2., # True value is 3.
        "center": {"value": [25.5,25.5], "locked":True},
        "I0": {"value": 1., "locked":True},
    },
    psf_upscale = 1.,
)

# Here we bind the PSF model to the galaxy model, this will add the psf_model␣
 ↪parameters to the galaxy_model
# object since it now depends on those.
live_galaxy_model.set_aux_psf(live_psf_model)

live_galaxy_model.initialize()

result = ap.fit.LM(live_galaxy_model, verbose = 1).fit()
result.update_uncertainty()
print(result.message)
```

```
L: 1.0
---------init---------
LM loss: 1370.6501183950186
L: 1.0
---------iter---------
LM loss: 1655.500357036183
reject
L: 11.0
---------iter---------
LM loss: 1298.657289797574
accept
L: 1.2222222222222223
---------iter---------
```

```
LM loss: 791.320423379912
accept
L: 0.1358024691358025
---------iter---------
LM loss: 14.657767169627471
accept
L: 0.015089163237311388
---------iter---------
LM loss: 5.120436840695977
accept
L: 0.0016765736930345987
---------iter---------
LM loss: 1.0184591243424452
accept
L: 0.00018628596589273318
---------iter---------
LM loss: 1.012427303968397
accept
L: 2.069844065474813e-05
---------iter---------
LM loss: 1.0124263177613597
accept
L: 2.299826739416459e-06
---------iter---------
LM loss: 1.0124263160676688
accept
success
```

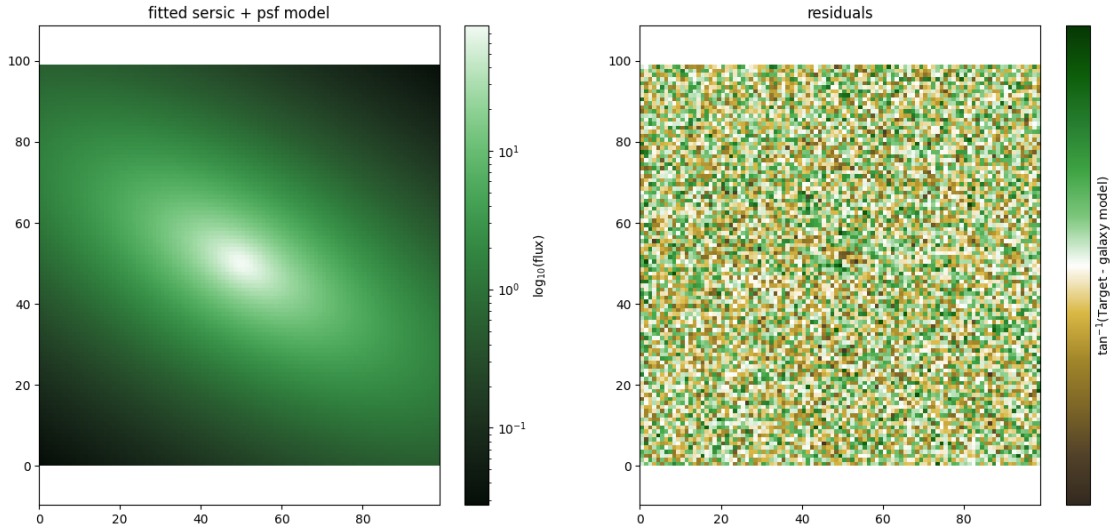
[6]:
```
print("fitted n for moffat PSF: ", live_galaxy_model["psf:n"].value.item(), "we␣
  ↪were hoping to get 2!")
print("fitted Rd for moffat PSF: ", live_galaxy_model["psf:Rd"].value.item(),␣
  ↪"we were hoping to get 3!")
print(live_galaxy_model.parameters)
```

```
fitted n for moffat PSF:  1.9954904924658476 we were hoping to get 2!
fitted Rd for moffat PSF:  3.0543224547473926 we were hoping to get 3!
Parameter Group: galaxy model
center: [50.04093493305835, 50.03636685305573] +- [0.0031088409908010908,
0.0023167787101885775] [arcsec]
q: 0.3982476939805446 +- 0.0015903931550282533 [b/a, (0.0, 1.0)]
PA: 1.0473316173110359 +- 0.0003600837185808103 [radians, (0.0,
3.141592653589793), cyclic]
n: 2.0027161574067427 +- 0.011704122854731658 [none, (0.36, 8.0)]
Re: 24.873380674262975 +- 0.042148600785378534 [arcsec, (0.0, 'None')]
Ie: 1.0055522682296427 +- 0.0016198966965689898 [log10(flux/arcsec^2)]
center: [25.5, 25.5] +- None [arcsec, locked]
n: 1.9954904924658476 +- 0.02452352406514529 [none, (0.1, 10.0)]
Rd: 3.0543224547473926 +- 0.02392361101783449 [arcsec, (0.0, 'None')]
```

```
I0: 1.0 +- None [log10(flux/arcsec^2), locked]
```

This is truly remarkable! With no stars available we were still able to extract an accurate PSF from the image! To be fair, this example is essentially perfect for this kind of fitting and we knew the true model types (sersic and moffat) from the start. Still, this is a powerful capability in certain scenarios. For many applications (e.g. weak lensing) it is essential to get the absolute best PSF model possible. Here we have shown that not only stars, but galaxies in the field can be useful tools for measuring the PSF!

[7]:
```
fig, ax = plt.subplots(1,2, figsize = (16,7))
ap.plots.model_image(fig, ax[0], live_galaxy_model)
ap.plots.residual_image(fig, ax[1], live_galaxy_model)
ax[0].set_title("fitted sersic + psf model")
ax[1].set_title("residuals")
plt.show()
```



There are regions of parameter space that are degenerate and so even in this idealized scenario the PSF model can get stuck. If you rerun the notebook with different random number seeds for pytorch you may find some where the optimizer "fails by immobility" this is when it get's stuck in the parameter space and can't find any way to improve the likelihood. In fact most of these "fail" fits do return really good values for the PSF model, so keep in mind that the "fail" flag only means the possibility of a truly failed fit. Unfortunatly, detecting convergence is hard.

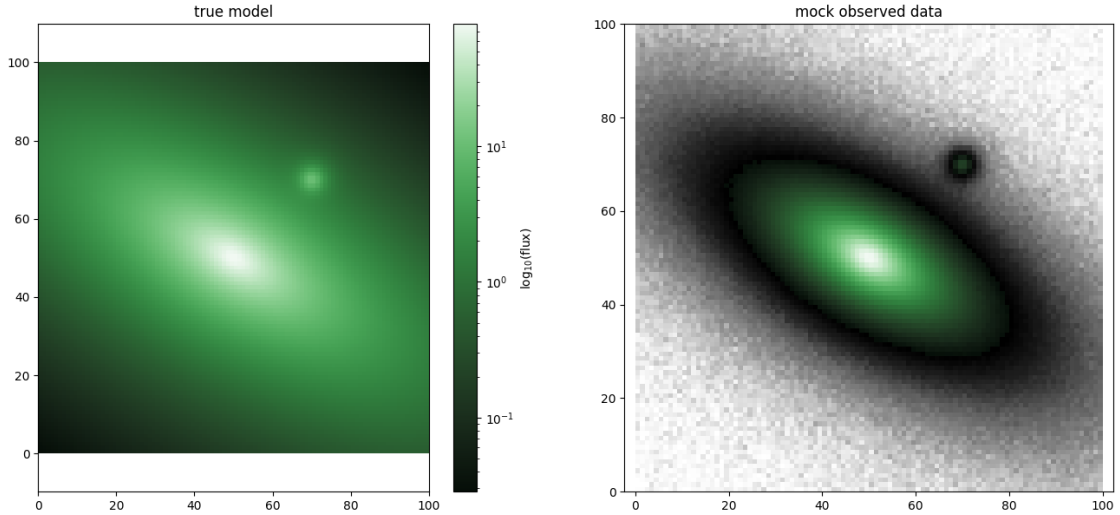
2.1 PSF fitting with a bad star

Fitting a PSF to a galaxy is perhaps not the most stable way to get a good model. However, there is a very common situation where this kind of fitting is quite helpful. Consider the scenario that there is a star, but it is not very bright and it is right next to a galaxy. Now we need to model the galaxy and the star simultaneously, but the galaxy should be convolved with the PSF for the fit to be stable (otherwise you'll have to do several iterations to converge). If there were many stars you

could perhaps just stack a bunch of them and hope the average is close enough, but in this case we don't have many to work with so we need to squeeze out as much statistical power as possible.

[8]:
```
# Lets make some data that we need to fit

true_psf2 = ap.utils.initialize.moffat_psf(
    2., # n                                  !!!!! Take note, we want to get n =␣
 ↪2. !!!!!!
    3., # Rd                                 !!!!! Take note, we want to get Rd =␣
 ↪3.!!!!!!
    51, # pixels
    1.  # pixelscale
)

target2 = ap.image.Target_Image(
    data = torch.zeros(100,100),
    pixelscale = 1.,
    psf = true_psf,
)

true_galaxy2 = ap.models.AutoPhot_Model(
    name = "true galaxy",
    model_type = "sersic galaxy model",
    target = target2,
    parameters = {
        "center": [50.,50.],
        "q": 0.4,
        "PA": np.pi/3,
        "n": 2,
        "Re": 25,
        "Ie": 1,
    },
    psf_mode = "full",
)
true_star2 = ap.models.AutoPhot_Model(
    name = "true star",
    model_type = "moffat star model",
    target = target2,
    parameters = {
        "center": [70,70],
        "n": 2,
        "Rd": 3,
        "I0": 1.,
    },
)
true_model2 = ap.models.AutoPhot_Model(
    name = "true model",
```

```
    model_type = "group model",
    target = target2,
    models = [true_galaxy2, true_star2],
)

# use the true model to make some data
sample = true_model2()
torch.manual_seed(1618033988)
target2.data = sample.data + torch.normal(torch.zeros_like(sample.data), 0.1)
target2.variance = 0.01 * torch.ones_like(sample.data)

fig, ax = plt.subplots(1,2, figsize = (16,7))
ap.plots.model_image(fig, ax[0], true_model2)
ap.plots.target_image(fig, ax[1], target2)
ax[0].set_title("true model")
ax[1].set_title("mock observed data")
plt.show()
```

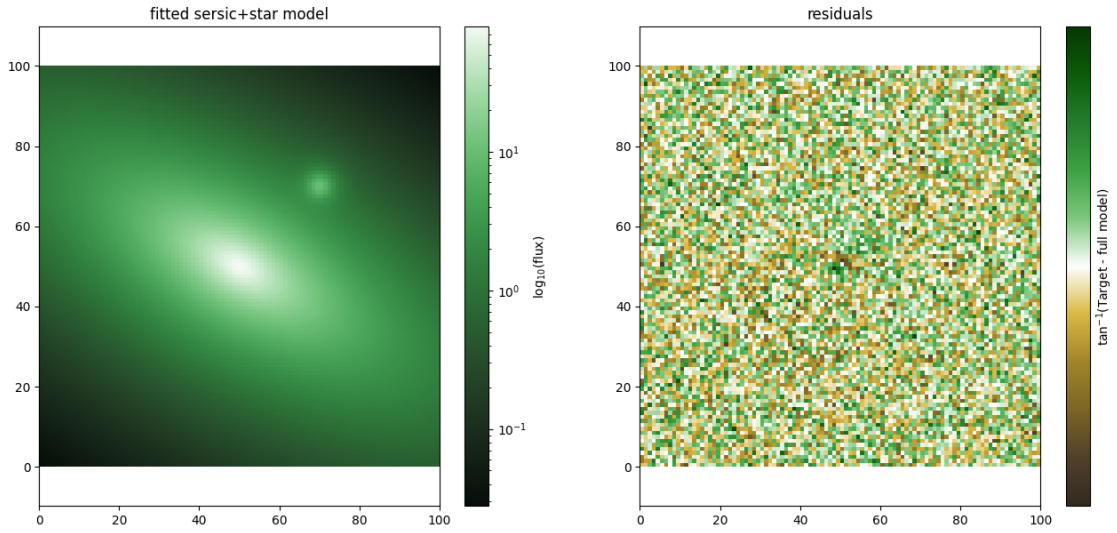


[9]:
```
# Now we will try and fit the data

# Here we set up a sersic model for the galaxy
galaxy_model2 = ap.models.AutoPhot_Model(
    name = "galaxy model",
    model_type = "sersic galaxy model",
    target = target,
    psf_mode = "full",
    psf_subpixel_shift = False,
)
```

```
psf_model2 = ap.models.AutoPhot_Model(
    name = "psf",
    model_type = "moffat star model",
    target = target2,
    window = [[0,51],[0,51]],
    parameters = {
        "n": 1., # True value is 2.
        "Rd": 2., # True value is 3.
        "center": {"value": [25.5,25.5], "locked":True},
        "I0": {"value": 1., "locked":True},
    },
    psf_upscale = 1.,
)

# Here we bind the PSF model to the galaxy model, this will add the psf_model␣
 ↪parameters to the galaxy_model
# object since it now depends on those.
galaxy_model2.set_aux_psf(psf_model2)

# Let AutoPhot determine its own intial parameters, so it has to start with␣
 ↪whatever it decides automatically,
# just like a real fit.
galaxy_model2.initialize()

star_model2 = ap.models.AutoPhot_Model(
    name = "star model",
    model_type = "moffat star model",
    target = target2,
    parameters = {
        "center": [72,68], # start the star in roughly the right location
    },
)

star_model2.initialize()

star_model2.add_equality_constraint(psf_model2, ["n", "Rd"])

full_model2 = ap.models.AutoPhot_Model(
    name = "full model",
    model_type = "group model",
    models = [galaxy_model2, star_model2],
    target = target2,
)

result = ap.fit.LM(full_model2, verbose = 1).fit()
print(result.message)
```

```
L: 1.0
---------init---------
LM loss: 1347.8559368224305
L: 1.0
---------iter---------
LM loss: 1737.3475714639603
reject
L: 11.0
---------iter---------
LM loss: 1279.2213988756682
accept
L: 1.2222222222222223
---------iter---------
LM loss: 795.9916667708957
accept
L: 0.1358024691358025
---------iter---------
LM loss: 18986.30658155843
reject
L: 1.4938271604938274
---------iter---------
LM loss: 24.98687279747573
accept
L: 0.16598079561042525
---------iter---------
LM loss: 11.594887217458497
accept
L: 0.018442310623380583
---------iter---------
LM loss: 3.5967479429076334
accept
L: 0.0020491456248200647
---------iter---------
LM loss: 2.170085146797211
accept
L: 0.0002276828472022294
---------iter---------
LM loss: 1.9266808377335043
accept
L: 2.5298094133581044e-05
---------iter---------
LM loss: 1.9265349459681327
accept
L: 2.8108993481756715e-06
---------iter---------
LM loss: 1.9265349243974574
accept
success
```

[10]:
```
fig, ax = plt.subplots(1,2, figsize = (16,7))
ap.plots.model_image(fig, ax[0], full_model2)
ap.plots.residual_image(fig, ax[1], full_model2)
ax[0].set_title("fitted sersic+star model")
ax[1].set_title("residuals")
plt.show()
```



[11]:
```
print("fitted n for moffat PSF: ", galaxy_model2["psf:n"].value.item(), "we␣
 ↪were hoping to get 2!")
print("fitted Rd for moffat PSF: ", galaxy_model2["psf:Rd"].value.item(), "we␣
 ↪were hoping to get 3!")

print("---Note that we can just as well look at the star model parameters since␣
 ↪they are the same---")
print("fitted n for moffat PSF: ", star_model2["n"].value.item(), "we were␣
 ↪hoping to get 2!")
print("fitted Rd for moffat PSF: ", star_model2["Rd"].value.item(), "we were␣
 ↪hoping to get 3!")
```

```
fitted n for moffat PSF:  1.982517944196754 we were hoping to get 2!
fitted Rd for moffat PSF:  3.0305342258869135 we were hoping to get 3!
---Note that we can just as well look at the star model parameters since they
are the same---
fitted n for moffat PSF:  1.982517944196754 we were hoping to get 2!
fitted Rd for moffat PSF:  3.0305342258869135 we were hoping to get 3!
```

Note that the fitted moffat parameters aren't much better than they were earlier when we just fit the galaxy alone. This shows us that extended objects have plenty of constraining power when it comes to PSF fitting, all this information has previously been left on the table! It makes sense that the galaxy dominates the PSF fit here, while the star is very simple to fit, it has much less light

than the galaxy in this scenario so the S/N for the galaxy dominates. The reason this works really well is of course that the true data is in fact a sersic model, so we are working in a very idealized scenario. Real world galaxies are not necessarily well described by a sersic, so it is worthwhile to be cautious when doing this kind of fitting. Always make sure the results make sense before storming ahead with galaxy based PSF models, that said the payoff can be well worth it.

3 PSF fitting for faint stars

Sometimes there are stars available, but they are faint and it is hard to see how a reliable fit could be obtained. We have already seen how faint stars next to galaxies are still viable for PSF fitting. Now we will consider the case of isolated but faint stars. The trick here is that we have a second high resolution image, perhaps in a different band. To perform this fitting we will link up the two bands using joint modelling to constrain the star centers, this will constrain some of the parameters making it easier to fit a PSF model.

[ ]:
```
# Coming soon
```

4 PSF fitting for saturated stars

A saturated star is a bright star, and it's just begging to be used for modelling a PSF. There's just one catch, the highest signal to noise region is completely messed up and can't be used! Traditionally these stars are either ignored, or a two stage fit is performed to get an "inner psf" and an "outer psf" which are then merged. Why not fit the inner and outer PSFs all at once! This can be done with AutoPhot using parameter constraints and masking.

[ ]:
```
# Coming soon
```