Usage of the Galaxy Builder analysis package

This notebook runs through some example uses of the `gzbuilder_analysis` package (available on Github).

In this notebook we:

- take an example galaxy and extracts an aggregate model, including logarithmic spiral arms
- render volunteer models to identify the best individual classification
- further optimize this classification
- compare this fitted model it to an optimized version of the aggregate model.

[1]

```
%load_ext autoreload
%autoreload 2
```

Needed imports (in a verbose manner)

[2]

```
import json
import numpy as np
from copy import deepcopy
import matplotlib.pyplot as plt
from shapely.geometry import MultiPolygon
from descartes import PolygonPatch
from tqdm import tqdm
from sklearn.metrics import mean_squared_error
from IPython.display import display, update_display,
display_html, HTML

import gzbuilder_analysis.parsing as parsing
import gzbuilder_analysis.spirals as spirals
from gzbuilder_analysis.spirals.oo import Pipeline
import gzbuilder_analysis.aggregation as aggregation
from gzbuilder_analysis.aggregation import average_shape_helpers
as ash
import gzbuilder_analysis.rendering as rendering
import gzbuilder_analysis.fitting as fitting

import lib.galaxy_utilities as gu
```

Define the subject ID to work on

Papermill - Parametrized

```
[3]  subject_id = 20902040
```

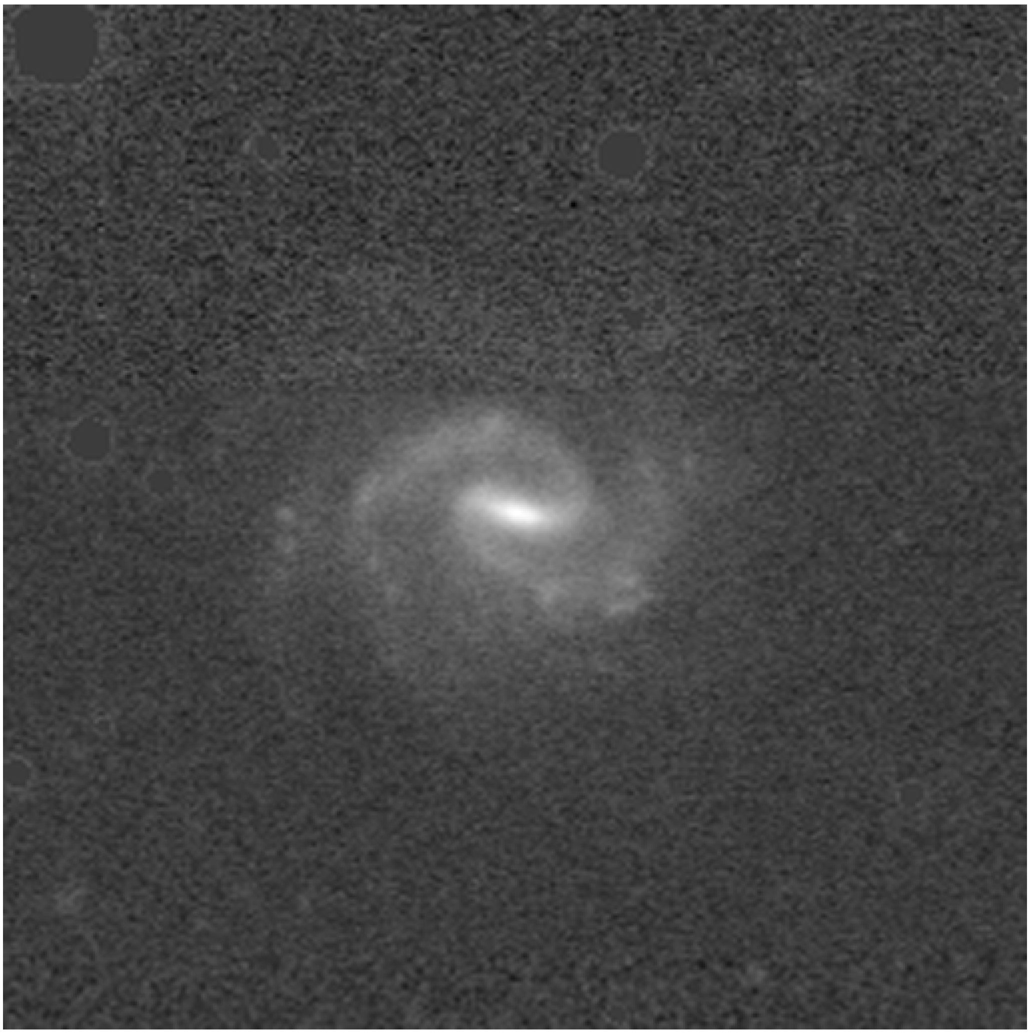
Grab the classifications for this galaxy, along with other needed metadate (originally obtained from either the NASA-Sloan Atlas, or SDSS skyserver, but can mostly be found inside the subject metadata)

```
[4]  cls = gu.classifications.query('subject_ids ==
     {}'.format(subject_id))
     drawn_arms = spirals.get_drawn_arms(cls)

     gal, angle = gu.get_galaxy_and_angle(subject_id)
     ba = gal['PETRO_BA90']
     im = gu.get_image(subject_id)
     psf = gu.get_psf(subject_id)
     diff_data = gu.get_diff_data(subject_id)
     pixel_mask = 1 - np.array(diff_data['mask'])[::-1]
     galaxy_data = np.array(diff_data['imageData'])[::-1]
     size_diff = diff_data['width'] / diff_data['imageWidth']

     # functions for plotting
     tv = lambda v: parsing.transform_val(v, np.array(im).shape[0],
     gal['PETRO_THETA'])
     ts = lambda v: parsing.transform_shape(v, galaxy_data.shape[0],
     gal['PETRO_THETA'])
     ts_a = lambda v: parsing.transform_shape(v, galaxy_data.shape[0],
     gal['PETRO_THETA'])
     imshow_kwargs = dict(cmap='gray', origin='lower', extent=[tv(0),
     tv(np.array(im).shape[0])]*2)
```

Image being classified



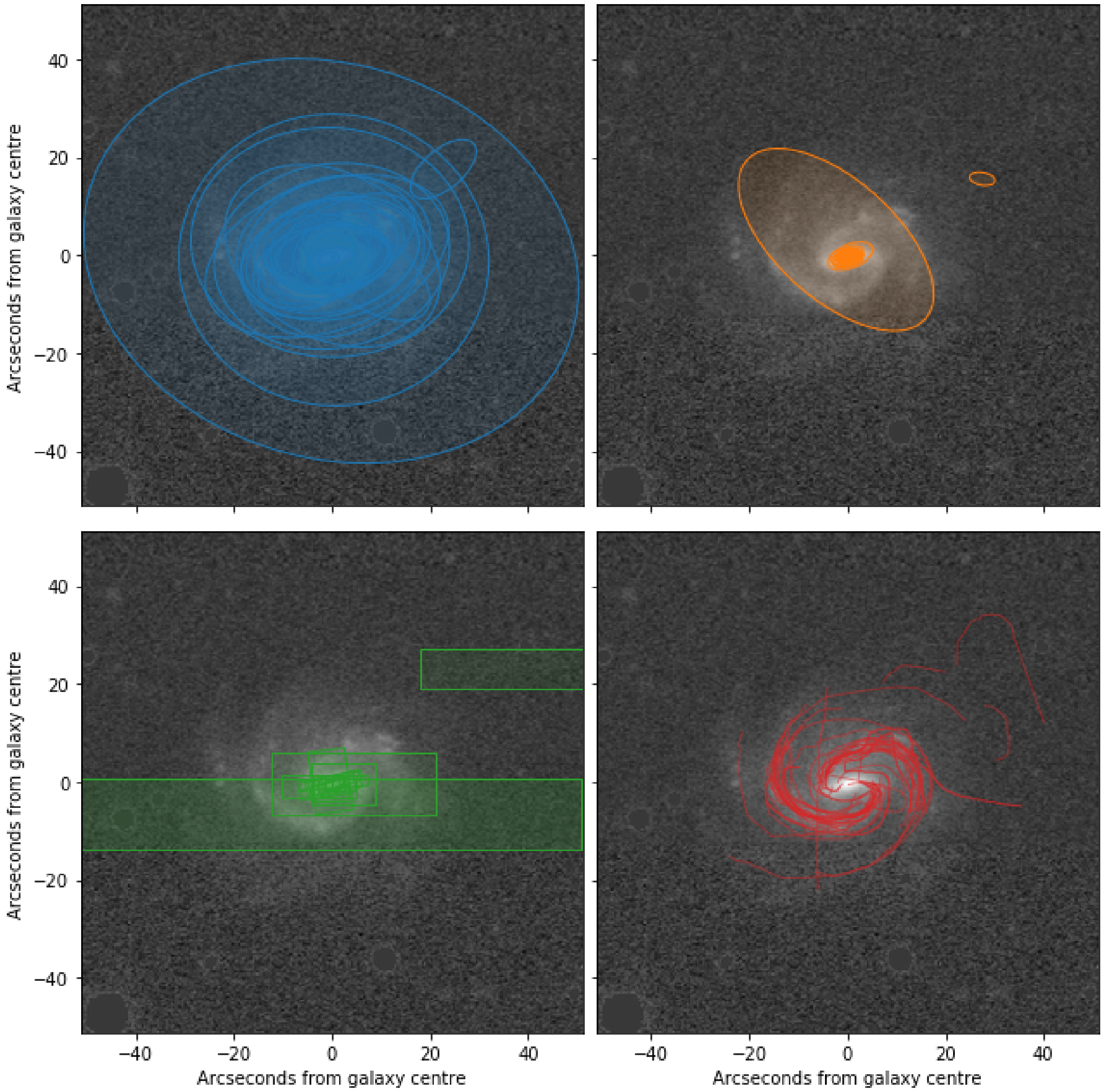
We'll quickly grab the annotations from these classifications, and use `gzbuilder_analysis.parsing` to scale them to Sloan pixels and put them in a more manageable format:

[6]

```
annotations = cls['annotations'].apply(json.loads)
models = annotations.apply(parsing.parse_annotation,
size_diff=size_diff)
```

Model calculation through clustering

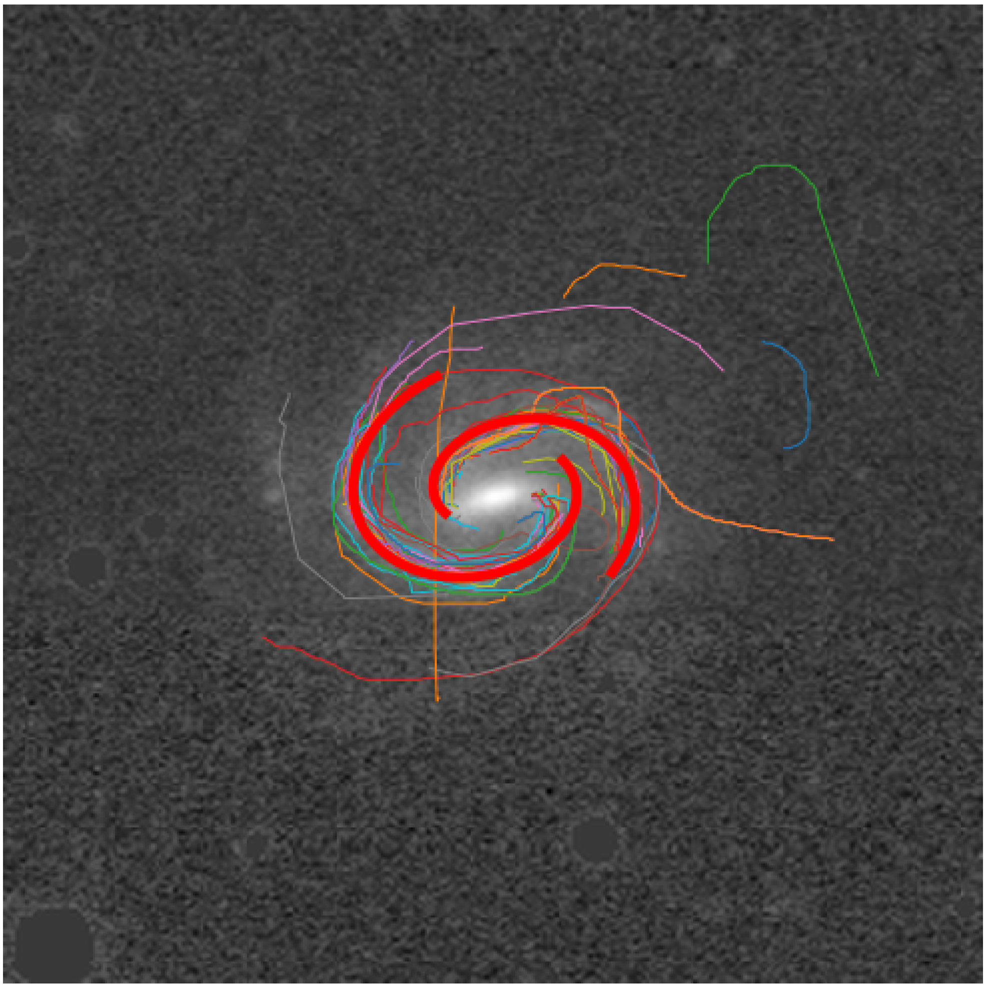
We can make use of unsupervised clustering to group drawn shapes and create a "model by consensus". First we'll visualize the drawn shapes to see what we're working with:



It's really easy to extract logarithmic spirals (if that's all you want)

[8]

```
p = spirals.oo.Pipeline(drawn_arms, phi=angle, ba=ba)
arms = p.get_arms()
```

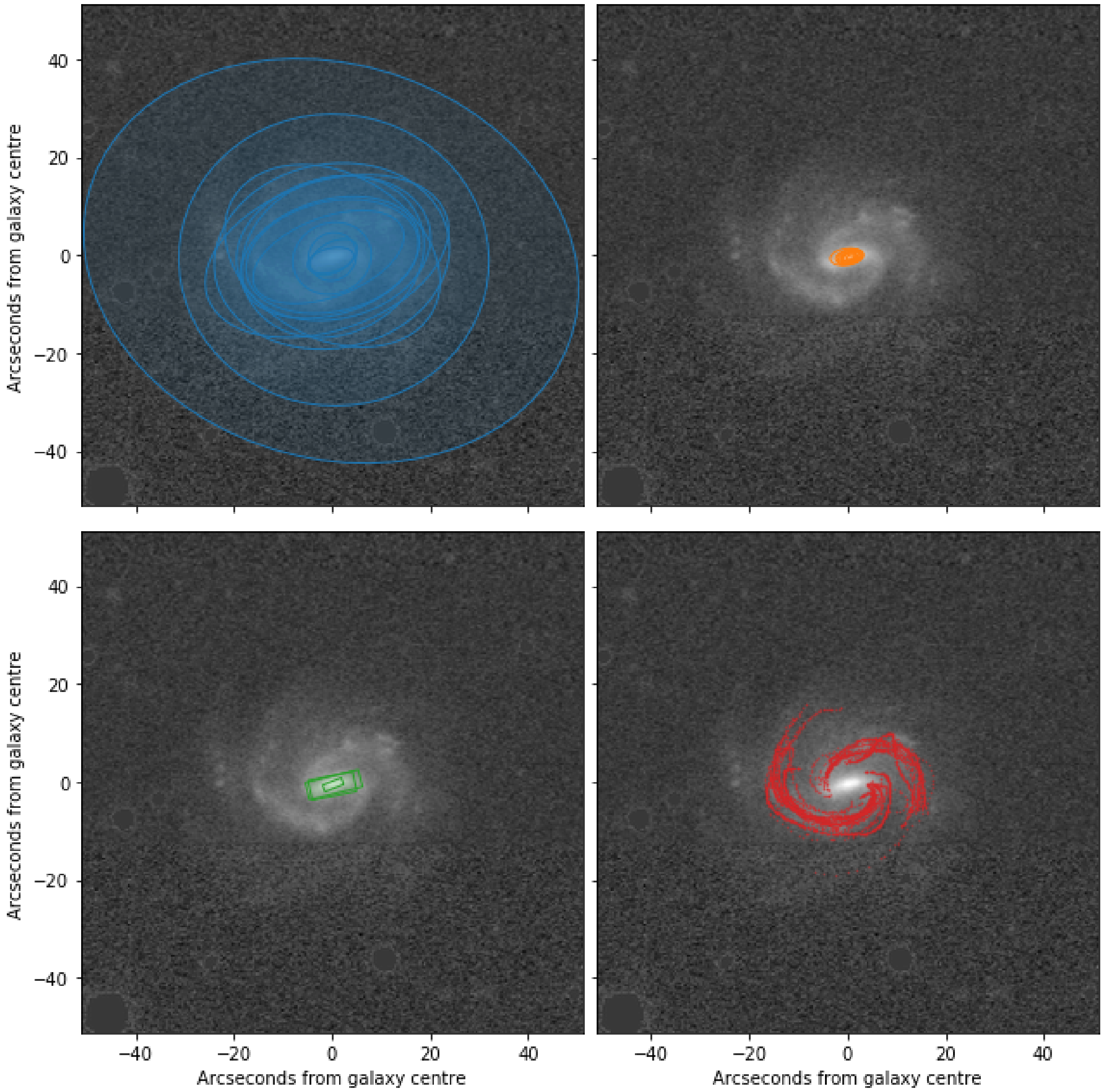


Alternatively, you can grab a complete aggregate model very easily (including spiral arms)

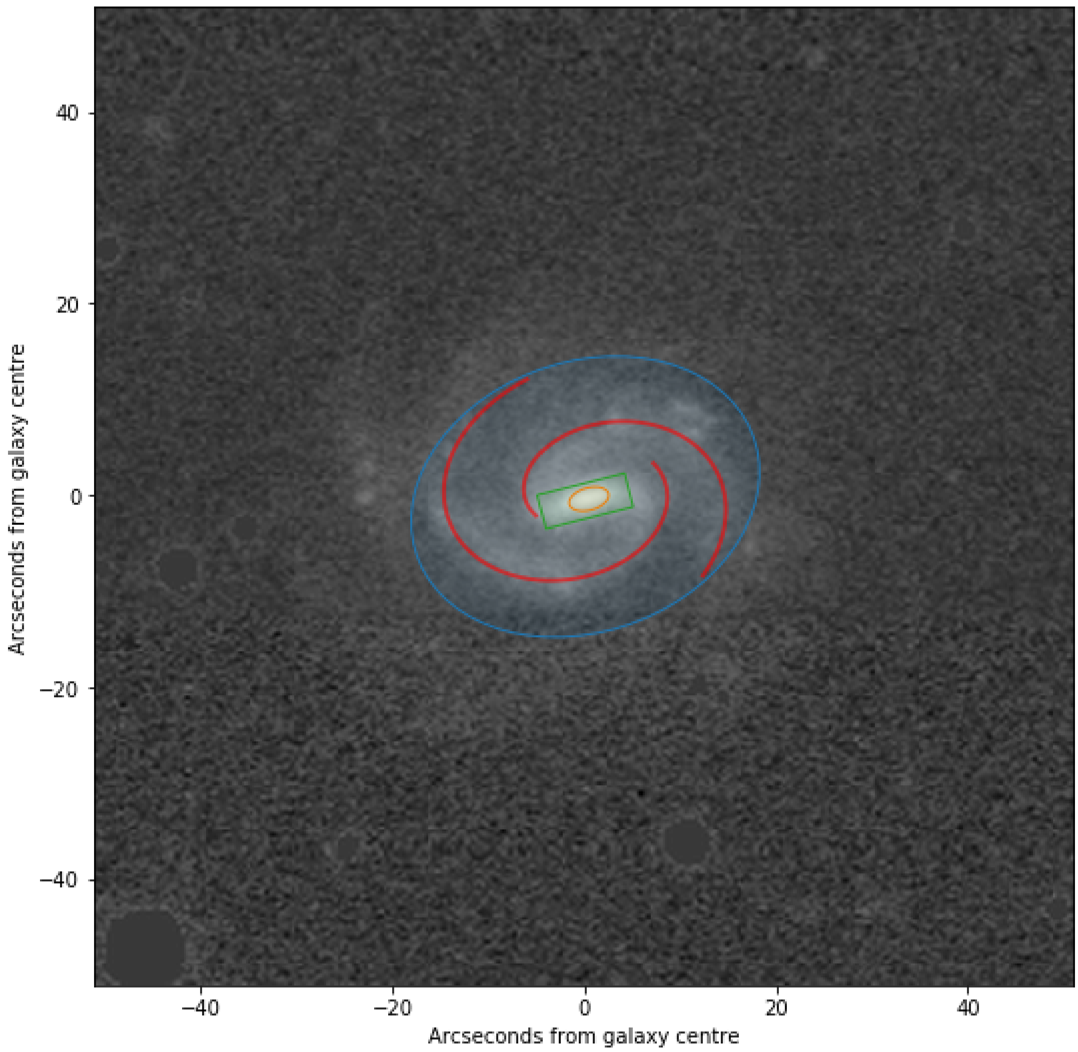
```
[10]  agg_res, masks, arms = aggregation.make_model(
          cls,
          gal,
          angle
      )
      agg_model = parsing.parse_aggregate_model(agg_res, size_diff)
```

```
[11]  annotations = cls['annotations'].apply(json.loads)
      models = annotations.apply(parsing.parse_annotation,
      size_diff=size_diff)
```

What components were identified as part of the cluster?



And what does the resulting aggregate model look like?



Best individual classification

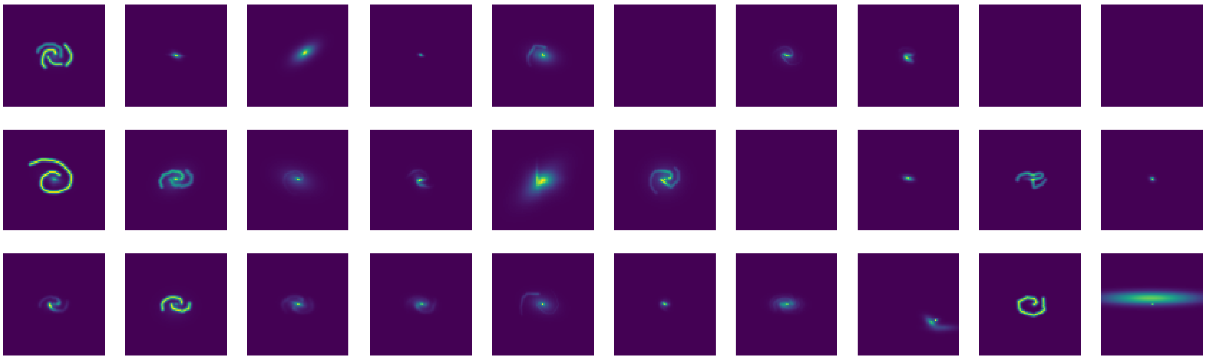
If you're not an aggregation kinda person, we could also identify the best individual classification:

[14]

```
annotations = cls['annotations'].apply(json.loads)
models = annotations.apply(parsing.parse_annotation,
size_diff=size_diff)

tqdm.pandas(desc='Rendering models')
rendered = models.progress_apply(rendering.calculate_model,
image_size=galaxy_data.shape[0], psf=psf)
```

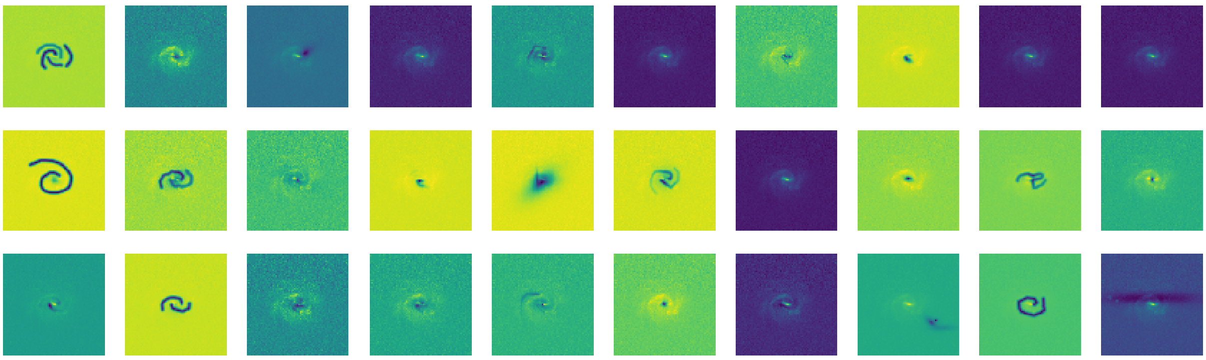
```
Rendering models: 100%|██████████| 32/32 [00:34<00:00,  1.75s/it]
```



Then calculate the difference between models and data and perform masking (unfortunately not trivial due to weird scalings present in the original code)

[16]
```
tqdm.pandas(desc='Calculating differences')
diffs = rendered.progress_apply(rendering.compare_to_galaxy,
args=(galaxy_data,),
                                pixel_mask=pixel_mask,
stretch=False)
```

```
Calculating differences: 100%|██████████| 32/32 [00:00<00:00,
560.39it/s]
```



Finally calculate the mean squared error for each model:

[18]
```
def loss(rendered_model):
    global galaxy_data, pixel_mask
    Y = rendered_model * pixel_mask
    return mean_squared_error(
        (rendered_model * pixel_mask).flatten(),
        0.8 * (galaxy_data * pixel_mask).flatten()
    )

losses = rendered.apply(loss)
```

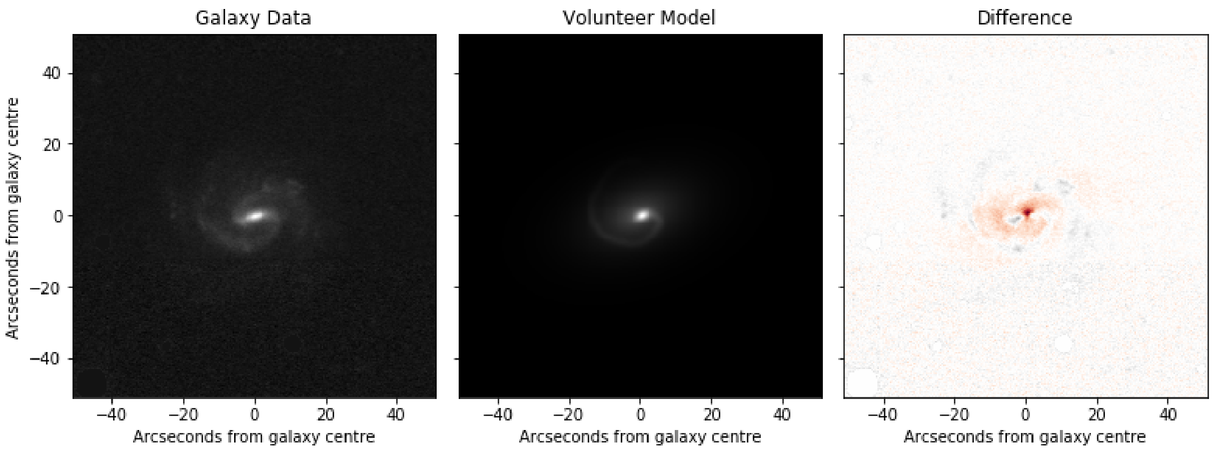
```
best = losses.idxmin()
```

```
[19]
print('Best model provided by', cls.loc[best].user_name)
md = fitting.Model(models.loc[best], galaxy_data, psf,
pixel_mask)
md
```

```
Best model provided by Kamphuisjes
```

	axRatio	c	i0	mu	n	rEff
disk	0.588732	2.0	0.13	[126.41048347949982, 128.45410412549973]	1.00	65.035871
bulge	0.516279	2.0	0.60	[129.43392097949982, 128.06217277050018]	0.78	6.007156
bar	0.355556	2.0	0.12	[129.68587410449982, 127.67024910449982]	0.73	3.708750

	falloff	i0	spread
Spiral number			
0	1.0	0.1	0.44
1	1.0	0.0	0.21



Optimization

We can now optimize the model parameters using `gzbulder_analysis.fitting`. Expect a two armed bulge + disc + bar galaxy to take upwards of three minutes to fit (ProFit we love you but you need spirals).

```
[21] mf = fitting.ModelFitter(models.loc[best], galaxy_data, psf,
     pixel_mask)
```

Running:
`fitted_model, res = mf.fit()`

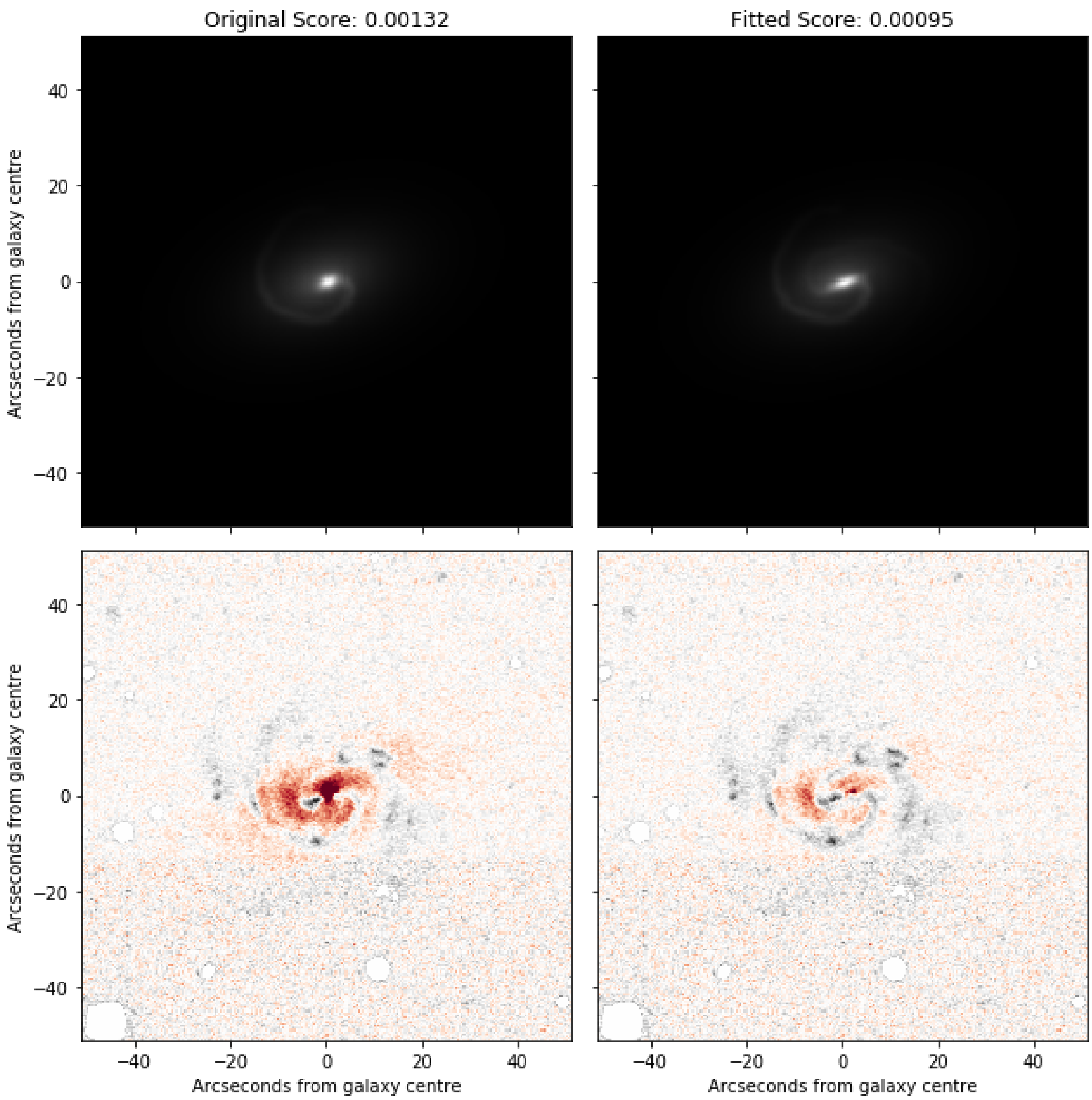
	axRatio	c	i0	mu	n
disk	0.588952	2.000000	0.104228	[126.41048347949982, 128.45410412549973]	1.000000
bulge	0.780869	2.000000	0.000000	[129.43392097949982, 128.06217277050018]	0.555409
bar	0.224649	2.086114	0.710014	[129.68587410449982, 127.67024910449982]	0.745745

	falloff	i0	spread
Spiral number			
0	0.861927	0.073528	0.481069
1	0.979016	0.025722	0.343841

```
Fitting model: 26it [03:23,  7.34s/it]
```

Successfuly completed fit

```
[23] fitted_rendered = rendering.calculate_model(fitted_model,
     image_size=galaxy_data.shape[0], psf=psf)
     fitted_difference = rendering.compare_to_galaxy(fitted_rendered,
     galaxy_data, pixel_mask=pixel_mask, stretch=False)
```



And what if we optimize the aggregate model? As we've had to guess at some of the values, we'll fit the model without spirals, then fit the model as a whole with the spirals starting at low brightness.

[25]

```
agg_model_nosp = deepcopy(agg_model)
spirals_removed = agg_model_nosp.pop('spiral')
agg_model_nosp['spiral'] = np.array([])

agg_mf_nosp = fitting.ModelFitter(agg_model_nosp, galaxy_data,
psf=psf,
                                  pixel_mask=pixel_mask)
```

Running:
`fitted_agg_nosp_model, agg_nosp_res = agg_mf_nosp.fit()`

	axRatio	c	i0	mu	n
disk	0.467799	2.000000	0.137766	[129.30788311713735, 128.65298894053245]	1.000000
bulge	0.290235	2.000000	0.412503	[130.18410465575397, 127.97919388809964]	1.080659
bar	0.719839	1.553503	0.080088	[129.24185156840736, 127.46517758694549]	0.336368

Spiral number

```
Fitting model: 22it [02:49,  8.23s/it]
```

Successfully completed fit

And now with low-brightness spirals:

```
[27]
def reset_spiral_intensity(s):
    points, params = s
    new_params = deepcopy(params)
    new_params['i0'] = 0.01
    return [points, new_params]

agg_model_with_spiral = {
  **deepcopy(fitted_agg_nosp_model),
  'spiral': [reset_spiral_intensity(s) for s in spirals_removed],
}
agg_mf = fitting.ModelFitter(agg_model_with_spiral, galaxy_data,
psf=psf,
                             pixel_mask=pixel_mask)
```

Running:

```
fitted_agg_model, agg_res = agg_mf.fit()
```

	axRatio	c	i0	mu	n
disk	0.472282	2.000000	0.118042	[129.30788311713735, 128.65298894053245]	1.000000
bulge	0.323483	2.000000	0.523696	[130.18410465575397, 127.97919388809964]	0.878029
bar	0.645650	1.527118	0.317539	[129.24185156840736, 127.46517758694549]	0.311778

	falloff	i0	spread

	falloff	i0	spread
Spiral number			
0	0.430284	0.063143	0.563996
1	0.790912	0.063776	0.592141

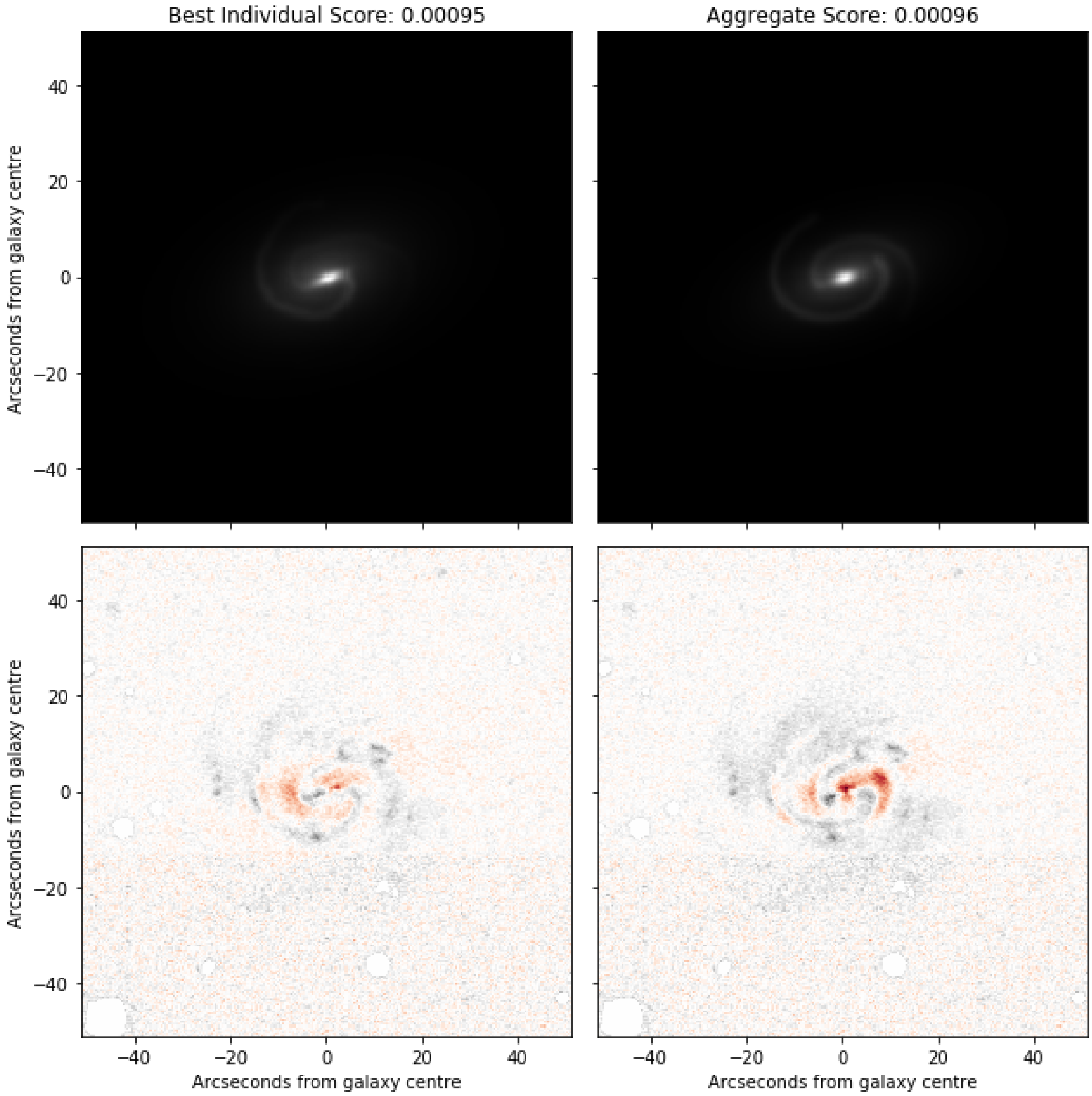
```
Fitting model: 14it [01:44,  8.42s/it]
```

Successfuly completed fit

We'll visualise the two models for easy comparison:

[29]

```
fitted_agg_rendered = rendering.calculate_model(fitted_agg_model,
image_size=galaxy_data.shape[0], psf=psf)
fitted_agg_difference =
rendering.compare_to_galaxy(fitted_agg_rendered, galaxy_data,
pixel_mask=pixel_mask, stretch=False)
```



Interestingly, whether the aggregate model outperforms the best individual classification often depends on which minima the fit gets stuck in - suggesting we should be using a fitting method more resistant to local minima, such as MCMC.

```
[31]
```