

# Clustering of drawn components in galaxy builder

First, say the jupyter magic words 🧙

```
[1] %matplotlib inline
    %load_ext autoreload
    %autoreload 2
```

Import the plethora of useful modules we'll need (including some we probably don't)

```
[2] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    from matplotlib.patches import Ellipse, Rectangle
    import json
    from sklearn.feature_extraction import DictVectorizer
    from panoptes_aggregation.extractors.shape_extractor import shape_extractor
    from panoptes_aggregation.extractors.utilities import annotation_by_task
    from panoptes_aggregation.reducers.shape_reducer_dbscan import shape_reducer_dbscan
    from panoptes_aggregation.reducers.shape_reducer_hdbscan import shape_reducer_hdbscan
    import galaxy_utilities as gu
    import wrangle_classifications as wc
    import render_galaxy as rg
```

Loading NSA catalog  
Loading Zooniverse classification dump  
Obtaining available frame montages

Which subject should we work on? (parametrised to allow batch running at a later date)

Papermill - Parametrized

```
[3] subject_id = 21096878
```

Implement a temporary way of reading in data till the zooniverse CSV export is fixed:

```
[4] with open('tmp_cls_dump.json') as f:
```

```

        classifications = json.load(f)
classifications_for_subject = [
    c for c in classifications
    if c['links']['subjects'][0] == str(subject_id)
]
print('Found {} classifications for subject_id {}'.format(
    len(classifications_for_subject),
    subject_id,
))
annotations_for_subject = [i['annotations'] for i in classifications_for_

```

Found 30 classifications for subject\_id 21096878

Grab the galaxy data (and classification data) for this galaxy

```

[5] print('Getting galaxy data')
gal, angle = gu.get_galaxy_and_angle(subject_id)
url = gu.getUrl(subject_id)
pic_array, deprojected_image = gu.get_image(gal, subject_id, angle)
# classifications = gu.classifications
# classifications_for_subject = classifications['subject_ids'] == subject
# annotations_for_subject = [
#     eval(foo) for foo in
#     classifications_for_subject
# ]

```

Getting galaxy data

Working on galaxy 21096878

- USING MONTAGED IMAGE

- Getting galaxy rotation

/Users/tlingard/anaconda3/lib/python3.7/site-

packages/skimage/transform/\_warps.py:110: UserWarning: Anti-aliasing will be enabled by default in skimage 0.15 to avoid aliasing artifacts when down-sampling images.

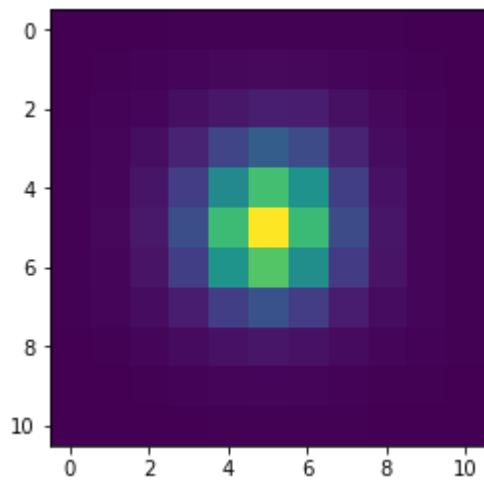
warn("Anti-aliasing will be enabled by default in skimage 0.15 to "

Grab the galaxy's PSF (from the data used in online model rendering)

```

[6] psf = gu.get_psf(subject_id)
plt.imshow(psf); None

```



Grab the difference JSON data for this subject (used later for rendering)

```
[7] diff_data = gu.get_image_data(subject_id)
```

Iterate through the annotations and grab the classifications for disks, bulges and bars:

```
[8] disks = [a[0] for a in annotations_for_subject]
bulges = [a[1] for a in annotations_for_subject]
bars = [a[2] for a in annotations_for_subject]
```

Let's have a look at these classifications!

```
[9] def plot_config(n):
    plt.subplot(n)
    plt.imshow(pic_array, origin='lower', cmap='gray')
    return plt.gca()

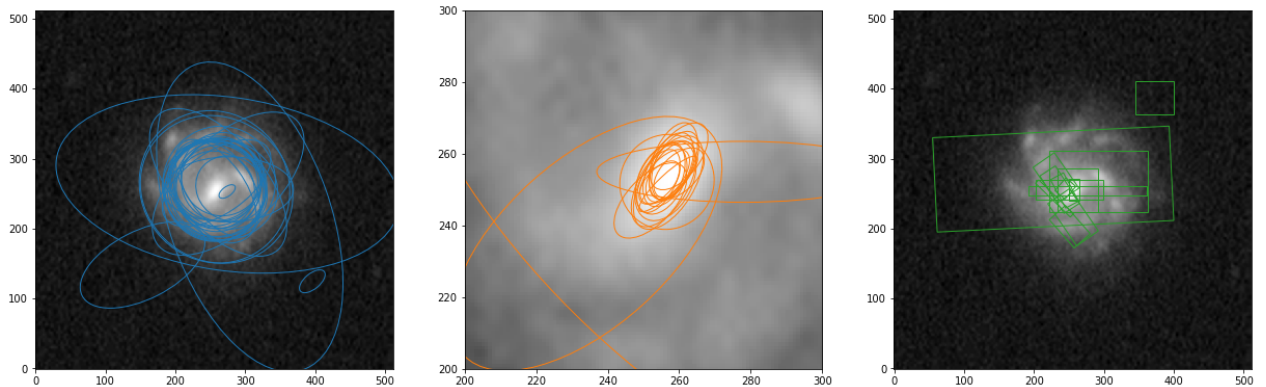
plt.figure(figsize=(21, 8))
for i, comp in enumerate((disks, bulges)):
    ax = plot_config(131 + i)
    for c_ in comp:
        try:
            c = c_['value'][0]['value'][0]
            ax.add_artist(Ellipse(
                xy=(c['x'], c['y']),
                width=c['rx'], height=c['ry'], angle=-c['angle'],
                ec='C{}'.format(i), fc='none',
            ))
        except IndexError: pass
    plt.xlim(200, 300); plt.ylim(200, 300)

ax = plot_config(133)
```

```

for bar in bars:
    try:
        b = bar['value'][0]['value'][0]
        r = ax.add_artist(Rectangle(
            xy=(b['x'], b['y']),
            width=b['width'], height=b['height'],
            angle=-b['angle'],
            ec='C2', fc='none',
        ))
    except IndexError: pass

```



We need to wrangle the classifications into a form the aggregation code accepts. This involves

- moving sliders to drawing sub-tasks
- move all shapes to the zero-th frame
- scale the radius according to the radius slider
- move the drawn shape out of the combo task, so it's on its own

This is all done using the `convert_shape` function inside `wrangle_classifications.py`

```

[10] converted_disks = [wc.convert_shape(d) for d in disks]
converted_bulges = [wc.convert_shape(d) for d in bulges]
convertedBars = [wc.convert_shape(d, bar=True) for d in bars]

```

Okay, let's have a look at what the edited disk shapes look like!

```

[11] plt.figure(figsize=(15, 7))
ax = plot_config(121)
for d_ in disks:
    try:
        d = d_['value'][0]['value'][0]
        e = Ellipse(
            xy=(d['x'], d['y']),
            width=d['rx'], height=d['ry'], angle=-d['angle'],
            ec='C0', fc='none',
        )
        ax.add_artist(e)

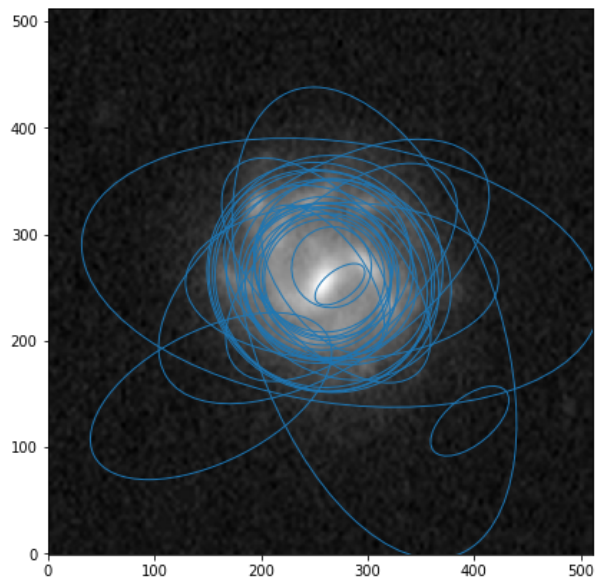
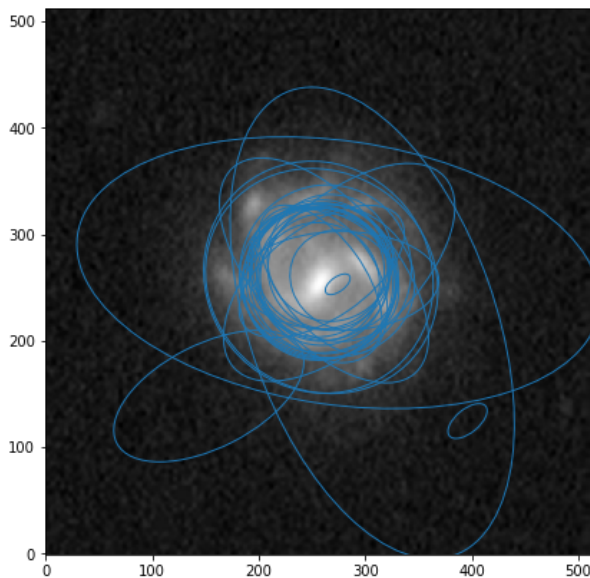
```

```

except IndexError: pass

ax = plot_config(122)
for d_ in converted_disks:
    try:
        d = d_['value'][0]
        e = Ellipse(
            xy=(d['x'], d['y']),
            width=d['rx'], height=d['ry'], angle=-d['angle'],
            ec='C0', fc='none',
        )
        ax.add_artist(e)
    except IndexError: pass

```



scikit-learn and pandas provide a handy way of doing more data exploration (not pursued here)

```

[12] # v = DictVectorizer(sparse=False)
      # flattened_disks = wc.sklearn_flatten([d['value'][0] for d in converted_
      # X = v.fit_transform(flattened_disks)
      # df_disk = pd.DataFrame(X, columns=v.get_feature_names()).drop(['frame',
      # df_disk.hist(); None

```

Define keyword arguments to be used by the aggregation extractor, and perform the reduction. More info can be found [here](#)

```

[13] kwargs_extractor = {
      'task': 'disk', 'shape': 'ellipse',
      'details': {'disk_tool0': [None, 'slider_extractor']},
    }
    extracted_disks = [
        shape_extractor(

```

```

        annotation_by_task({ 'annotations': [d] }),
        **kwargs_extractor
    )
    for d in converted_disks
]

```

Perform the aggregation reduction!

```

[14] kwargs_reducer = {
        'shape': 'ellipse',
        'details': {'disk_tool0': [None, 'slider_reducer']},
        'eps': 70, 'symmetric': True, 'min_samples': 5,
    }
    disk_clustering_result = shape_reducer_dbscan(
        extracted_disks,
        **kwargs_reducer,
    )['frame0']

```

```

[15] # disk_clustering_result

```

And plot the output clusters!

```

[16] plt.figure(figsize=(10, 10))
    ax = plt.gca()
    plt.imshow(pic_array, origin='lower', cmap='gray_r')
    for i in range(len(disk_clustering_result['disk_tool0_ellipse_rx'])):
        label = disk_clustering_result['disk_tool0_cluster_labels'][i]
        # ignore outliers
        if label < 0:
            continue
        try:
            e = Ellipse(
                xy=(
                    disk_clustering_result['disk_tool0_ellipse_x'][i],
                    disk_clustering_result['disk_tool0_ellipse_y'][i]
                ),
                width=disk_clustering_result['disk_tool0_ellipse_rx'][i],
                height=disk_clustering_result['disk_tool0_ellipse_ry'][i],
                angle=-disk_clustering_result['disk_tool0_ellipse_angle'][i],
                fc='none', ec='C{}'.format((label) % 10),
                alpha=0.5,
            )
            ax.add_artist(e)
        except IndexError:
            pass
        except KeyError as e:
            print(d)

```

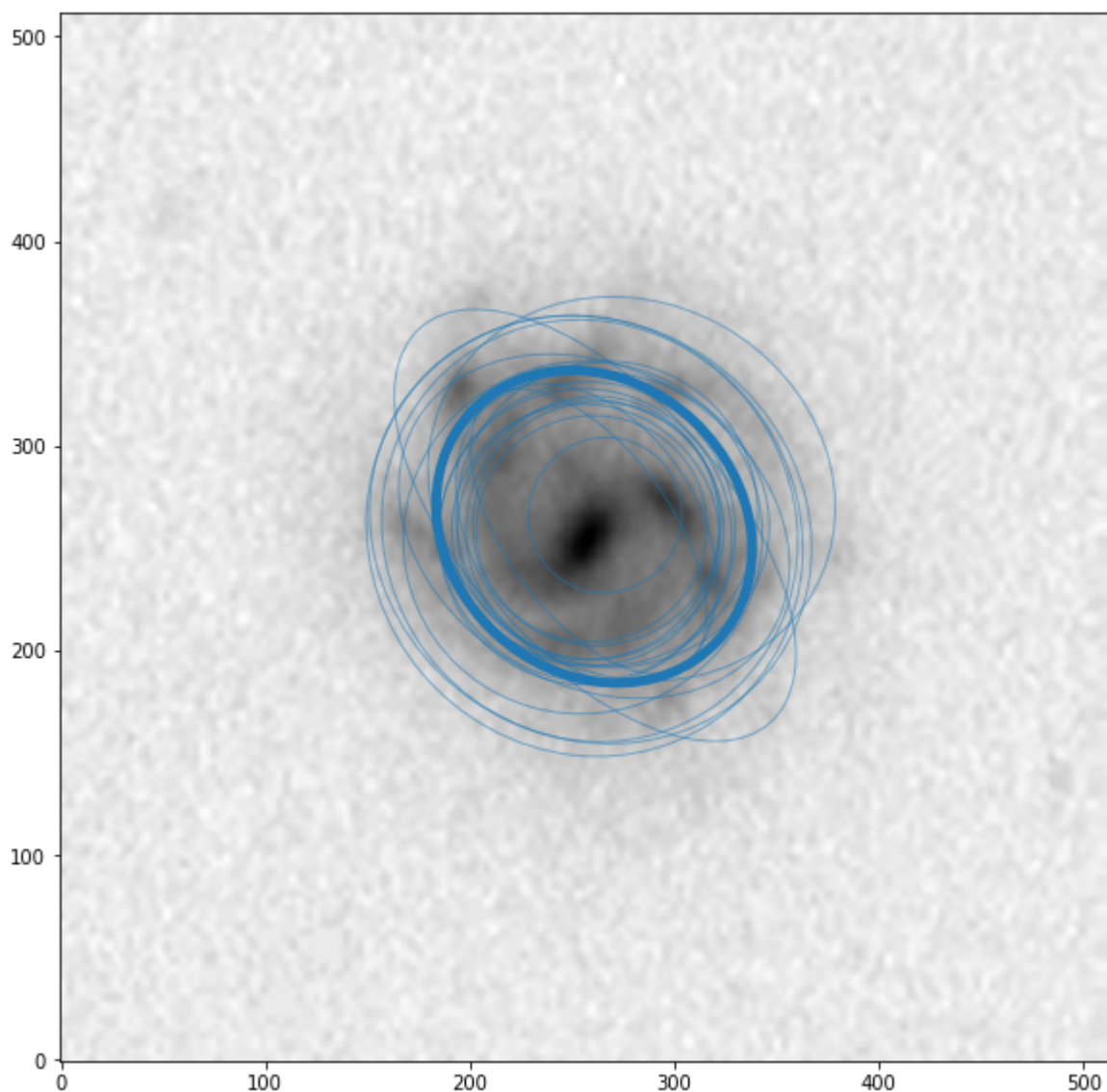
```

        raise(e)

    for label in np.unique(disk_clustering_result['disk_tool0_cluster_labels']):
        if label > -1:
            final_disk = Ellipse(
                xy=(disk_clustering_result['disk_tool0_clusters_x'][label], d
                width=disk_clustering_result['disk_tool0_clusters_rx'][label]
                height=disk_clustering_result['disk_tool0_clusters_ry'][label]
                angle=-disk_clustering_result['disk_tool0_clusters_angle'][la
                ec='C{}'.format(label),
                linewidth=5,
                fc='none',
            )

            ax.add_artist(final_disk)

```



```

[17] kwargs_extractor = {
    'task': 'bulge', 'shape': 'ellipse',
    'details': {'bulge_tool0': [None, 'slider_extractor', 'slider_extract
}
kwargs_reducer = {
    'shape': 'ellipse',

```

```

        'details': {'bulge_tool0': [None, 'slider_reducer', 'slider_reducer']
        'eps': 30, 'symmetric': True, 'min_samples': 5,
    }
    bulge_clustering_result = shape_reducer_dbscan(
        [
            shape_extractor(
                annotation_by_task({ 'annotations': [b] }),
                **kwargs_extractor
            )
            for b in converted_bulges
        ],
        **kwargs_reducer,
    )['frame0']

```

```

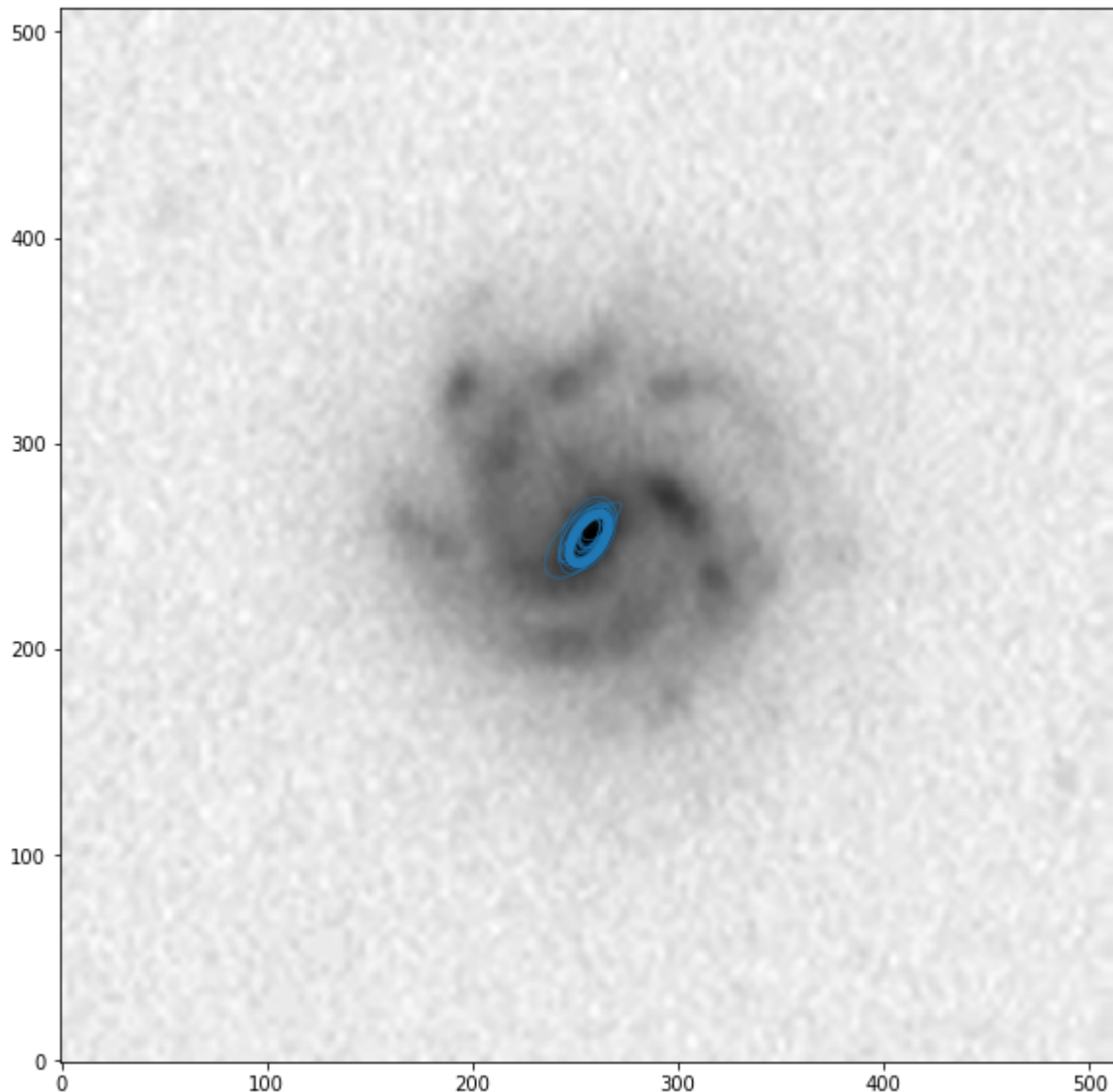
[18] plt.figure(figsize=(10, 10))
    ax = plt.gca()
    plt.imshow(pic_array, origin='lower', cmap='gray_r')
    for i in range(len(bulge_clustering_result['bulge_tool0_ellipse_rx'])):
        label = bulge_clustering_result['bulge_tool0_cluster_labels'][i]
        # ignore outliers
        if label < 0:
            continue
        try:
            e = Ellipse(
                xy=(
                    bulge_clustering_result['bulge_tool0_ellipse_x'][i],
                    bulge_clustering_result['bulge_tool0_ellipse_y'][i]
                ),
                width=bulge_clustering_result['bulge_tool0_ellipse_rx'][i],
                height=bulge_clustering_result['bulge_tool0_ellipse_ry'][i],
                angle=-bulge_clustering_result['bulge_tool0_ellipse_angle'][i]
                fc='none', ec='C{}'.format((label) % 10),
                alpha=0.5,
            )
            ax.add_artist(e)
        except IndexError:
            pass
        except KeyError as e:
            print(d)
            raise(e)

    for label in np.unique(bulge_clustering_result['bulge_tool0_cluster_label']):
        if label > -1:
            final_disk = Ellipse(
                xy=(bulge_clustering_result['bulge_tool0_clusters_x'][label],
                    bulge_clustering_result['bulge_tool0_clusters_rx'][label],
                    bulge_clustering_result['bulge_tool0_clusters_ry'][label],
                    bulge_clustering_result['bulge_tool0_clusters_angle'][label]
                    ec='C{}'.format(label),
                    linewidth=5,
                    fc='none',
                )

```



```
ax.add_artist(final_disk)
```



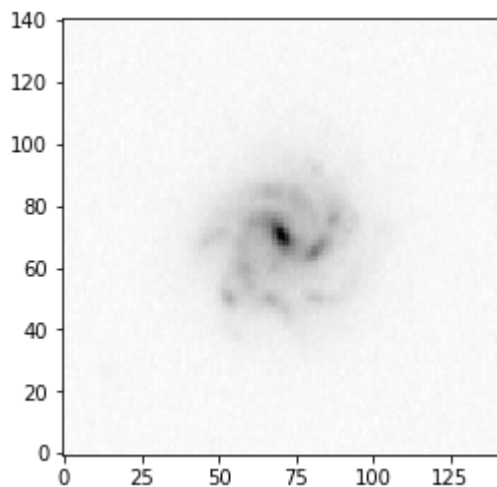
## Model Rendering

For simplicity, we'll only care about the "zeroth" cluster label.

Let's render the resulting galaxy and see what it looks like!

First, we need to transform our object back from Zooniverse image space to the original data space (512x512 image to 141x141)

```
[19] diff_data = gu.get_image_data(subject_id)
      galaxy_data_flipped = np.array(diff_data['imageData'])
      plt.imshow(galaxy_data_flipped, cmap='gray_r', origin='lower')
      None
```



It's worth noting that the y-axis is flipped from the image used before (which is the image shown to volunteers).

```
[20] galaxy_data = galaxy_data_flipped[::-1]
      size_diff = diff_data['width'] / diff_data['imageWidth']
```

Now we'll compile some parameters needed for model rendering:

First the Disk component

```
[21] rEff_disk = max(
      disk_clustering_result['disk_tool0_clusters_rx'][0],
      disk_clustering_result['disk_tool0_clusters_ry'][0],
    )
    axRatio_disk = disk_clustering_result['disk_tool0_clusters_rx'][0] \
        / disk_clustering_result['disk_tool0_clusters_ry'][0]

    roll_disk = disk_clustering_result['disk_tool0_clusters_angle'][0] \
        + (0 if axRatio_disk > 1 else 90)
    axRatio_disk = (1 / axRatio_disk if axRatio_disk > 1 else axRatio_disk)

[22] disk_to_render = {
      'mu': np.array((
        disk_clustering_result['disk_tool0_clusters_x'][0], disk_clustering_result['disk_tool0_clusters_y'][0]
      )) * size_diff,
      'roll': np.deg2rad(roll_disk),
      'rEff': rEff_disk * size_diff,
      'axRatio': axRatio_disk,
      'c': 2,
      'i0': disk_clustering_result['disk_tool0_clusters_details'][0][1]['sl'],
      'n': 1
    }
    disk_to_render
```

```
{'mu': array([71.7746948 , 71.73727175]),
 'roll': 0.7499679795819634,
 'rEff': 45.26404833682755,
 'axRatio': 0.8640327252861387,
 'c': 2,
 'i0': 0.19894736842105265,
 'n': 1}
```

Then the bulge component

```
[23] rEff_bulge = max(
        bulge_clustering_result['bulge_tool0_clusters_rx'][0],
        bulge_clustering_result['bulge_tool0_clusters_ry'][0],
    )
axRatio_bulge = bulge_clustering_result['bulge_tool0_clusters_rx'][0] \
    / bulge_clustering_result['bulge_tool0_clusters_ry'][0]

roll_bulge = bulge_clustering_result['bulge_tool0_clusters_angle'][0] \
    + (0 if axRatio_bulge > 1 else 90)
axRatio_bulge = (1 / axRatio_bulge if axRatio_bulge > 1 else axRatio_bulge)
```

```
[24] bulge_to_render = {
    'mu': np.array((
        bulge_clustering_result['bulge_tool0_clusters_x'][0], bulge_clust
    )) * size_diff,
    'roll': np.deg2rad(roll_bulge),
    'rEff': rEff_bulge * size_diff,
    'axRatio': axRatio_bulge,
    'c': 2,
    'i0': bulge_clustering_result['bulge_tool0_clusters_details'][0][1]['
    'n': bulge_clustering_result['bulge_tool0_clusters_details'][0][2]['s
}
bulge_to_render
```

```
{'mu': array([70.66709612, 69.85491584]),
 'roll': 2.1522178605117674,
 'rEff': 7.529863827595223,
 'axRatio': 0.5116509910683488,
 'c': 2,
 'i0': 0.7335294117647059,
 'n': 1.3852941176470586}
```

```
[25] disk_arr = rg.sersic_comp(disk_to_render, image_size=diff_data['width'],
                             oversample_n=5)

rg.plot_model(disk_arr, psf, galaxy_data)

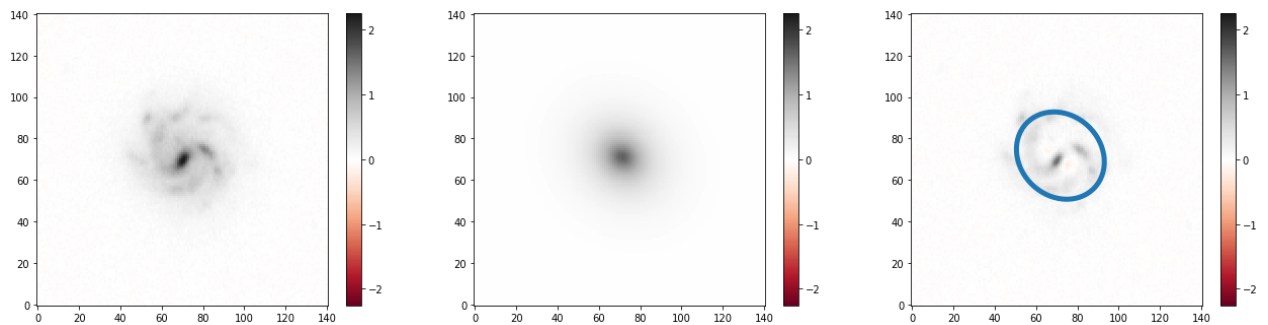
final_disk_kwargs = {
    'xy': disk_to_render['mu'],
```

```

        'width': disk_to_render['rEff'] * disk_to_render['axRatio'],
        'height': disk_to_render['rEff'],
        'angle': 90 - np.rad2deg(disk_to_render['roll']),
        'ec': 'C0',
        'linewidth': 5,
        'fc': 'none',
    }
    plt.gca().add_artist(Ellipse(**final_disk_kwargs))

```

<matplotlib.patches.Ellipse at 0x1c3dafd080>



```

[26] bulge_arr = rg.sersic_comp(bulge_to_render, image_size=diff_data['width']
                             oversample_n=5)

```

```

rg.plot_model(bulge_arr, psf, galaxy_data)

```

```

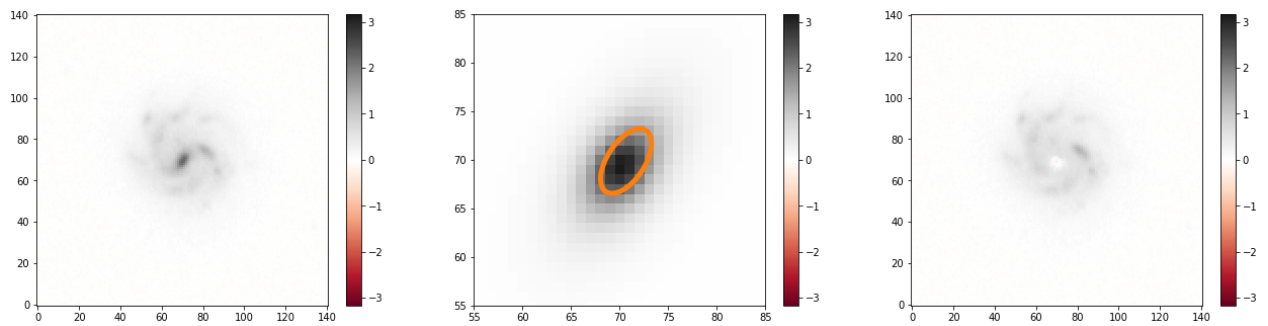
final_bulge_kwargs = {
    'xy': bulge_to_render['mu'],
    'width': bulge_to_render['rEff'] * bulge_to_render['axRatio'],
    'height': bulge_to_render['rEff'],
    'angle': 90 - np.rad2deg(bulge_to_render['roll']),
    'ec': 'C1',
    'linewidth': 5,
    'fc': 'none',
}
plt.subplot(132, label='model-data')
plt.gca().add_artist(Ellipse(**final_bulge_kwargs))
zoom_to = (55, 85)
plt.xlim(*zoom_to)
plt.ylim(*zoom_to)

```

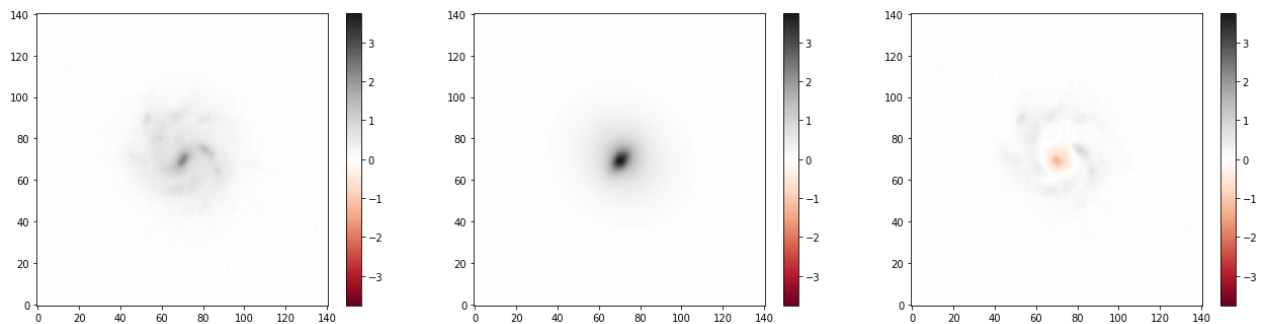
/Users/tlingard/anaconda3/lib/python3.7/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.

```
warnings.warn(message, mplDeprecation, stacklevel=1)
```

```
(55, 85)
```



```
[27] difference_data = rg.plot_model(disk_arr + bulge_arr, psf, galaxy_data)
# plt.gca().add_artist(Ellipse(**final_disk_kwargs))
# plt.gca().add_artist(Ellipse(**final_bulge_kwargs))
```



```
[29] from mpl_toolkits.mplot3d import Axes3D

Z = difference_data

X, Y = np.meshgrid(*[np.linspace(0, 1, difference_data.shape[0])]*2)
# Create a surface plot and projected filled contour plot under it.
fig = plt.figure(figsize=(10, 10))
ax = fig.gca(projection='3d')
ax.plot_surface(X, Y, Z, rstride=5, cstride=5, linewidth=0.1, antialiased
               cmap='RdGy')

cset = ax.contourf(X, Y, Z, zdir='z', offset=-3, cmap='RdGy')

# Adjust the limits, ticks and view angle
ax.set_zlim(-3, 3)
ax.set_zticks(np.linspace(-3, 3, 5))
ax.view_init(25, -21)
```

