Experiment on recovering spiral models

> Q: *Can we accurately recover the best model to use to fit spiral arms?*

Method:

- Generate template polynomial spirals, and a log spiral

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

[2]
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from IPython.display import display
from gzbuilderspirals import xy_from_r_theta, theta_from_pa
from gzbuilderspirals.oo import Pipeline
from matplotlib.patches import Ellipse
```

Reproducibility matters:

[3]
```
np.random.seed(0)
```

We need some array to parametrise distance along spiral arm

[4]
```
t = np.linspace(1, 2*np.pi+1)
```

Define how to make a log spiral, a polynomial spiral and how to creats noisy "drawn arms" from a clean template arm.

[5]
```
NOISE_LEVEL = 4
```

[6]
```
def log_spiral(p, t=t, dt=0):
    a, b = p
    return a * np.exp(b * t), t+dt

def poly_spiral(p, t=t, dt=0):
    return np.add.reduce([
```

```
            t**(i + 1) * p[i] for i in range(len(p))
        ], axis=0) + 20, t + dt

def gen_noisy_arm_from_tpl(template, v=NOISE_LEVEL):
    ltp = np.clip(np.random.randn()+1, 0, 3)
    large_scale_noise = (np.sin(ltp*template[1]) - 0.5) * np.random.randn
    r_n = template[0] \
        + np.random.randn(len(template[0])) * v \
        + large_scale_noise * v/4
    return r_n, template[1] + np.random.randn() * v * 0.05

def translate_arm(arm):
    return np.array(arm) + 256.0
```

Make some ground-truth template arms we hope to recover

```
[7]
p1 = [20]
p2 = [0, 2.8]
p3 = [0, 0, 0.45]
p_log = [15, np.tan(np.deg2rad(20))]

template_1_0 = poly_spiral(p1)
template_1_1 = poly_spiral(p1, dt=np.pi)
template_2_0 = poly_spiral(p2)
template_2_1 = poly_spiral(p2, dt=np.pi)
template_3_0 = poly_spiral(p3)
template_3_1 = poly_spiral(p3, dt=np.pi)
template_4_0 = log_spiral(p_log)
template_4_1 = log_spiral(p_log, dt=np.pi)
```

And stack them in a more useful form

```
[8]
labels = ('k=1 spiral', 'k=2 spiral', 'k=3 spiral', 'log spiral')

stacked_templates = (
    (template_1_0, template_1_1),
    (template_2_0, template_2_1),
    (template_3_0, template_3_1),
    (template_4_0, template_4_1),
)

template_arms = [
    [
        translate_arm(xy_from_r_theta(*t0)).T,
        translate_arm(xy_from_r_theta(*t1)).T,
    ]
    for t0, t1 in stacked_templates
]
```

Make the noisy drawn arms from our pretend volunteers, N for each template

[9]

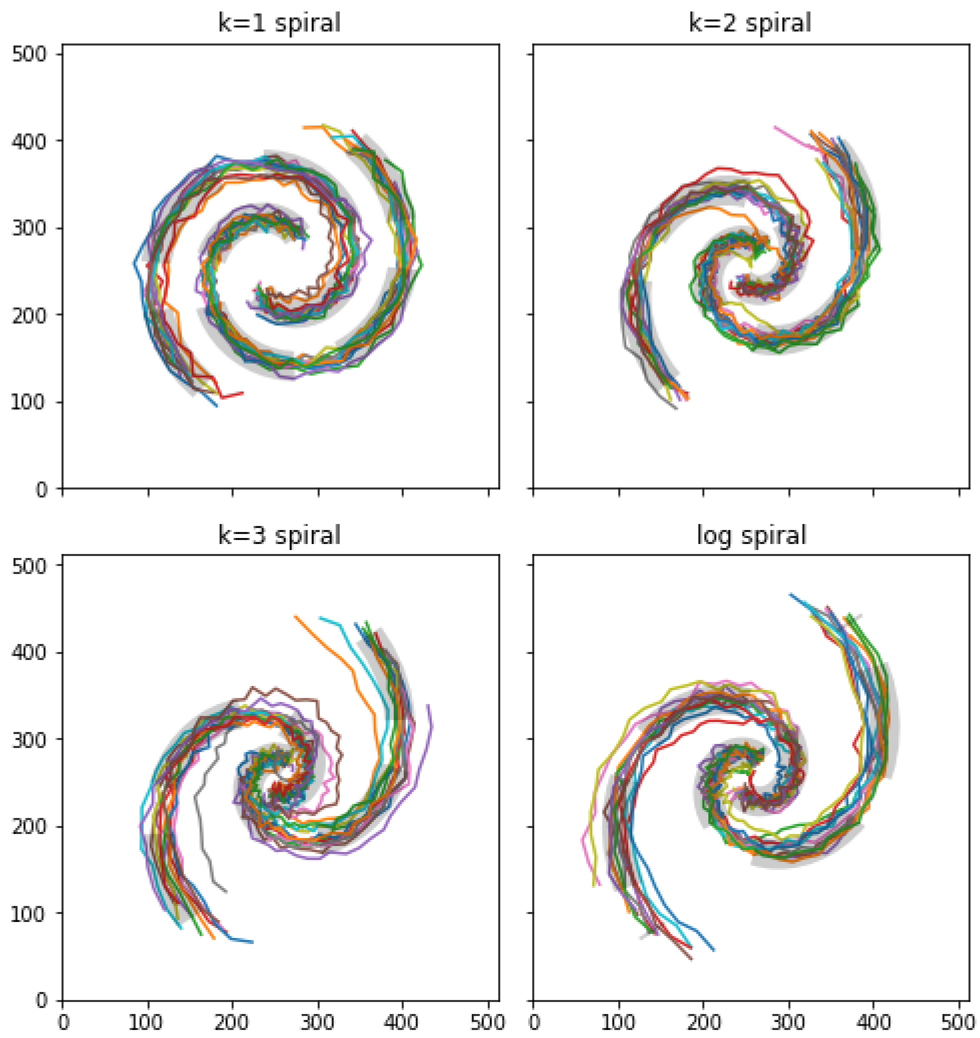
```
N = 13

drawn_arms = [
    [
        translate_arm(xy_from_r_theta(*gen_noisy_arm_from_tpl(t0))).T
        for i in range(N)
    ] + [
        translate_arm(xy_from_r_theta(*gen_noisy_arm_from_tpl(t1))).T
        for i in range(N)
    ]
    for t0, t1 in stacked_templates
]
```

Let's have a look at our beautifully noisy data 💥

[10]

```
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(7, 7.4), sharex=True,
axes = axes.reshape(-1)
for i, arms in enumerate(drawn_arms):
    plt.sca(axes[i])
    plt.xlim(0, 512)
    plt.ylim(0, 512)
    for arm in arms:
        plt.plot(*arm.T)
    for arm in template_arms[i]:
        plt.plot(*arm.T, 'k--', linewidth=15, alpha=0.2)
    plt.title(labels[i])
plt.tight_layout();
```



And now run everything through the `gzbuilderspirals` Pipeline interface!

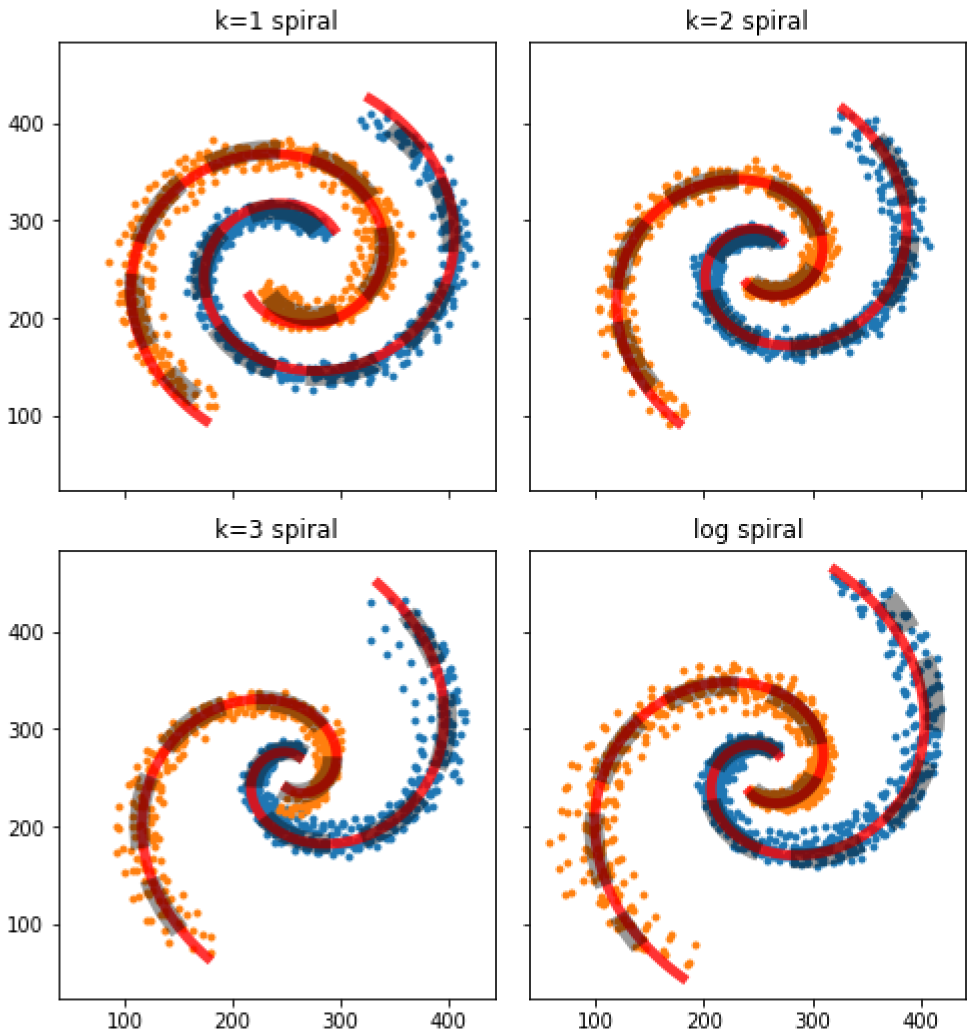
```
[11]  pipelines = [
          Pipeline(arms, phi=0, ba=1, image_size=512,
                   distances=None, parallel=True, bar_length=10)
          for arms in drawn_arms
      ]
```

```
[12]  arm_pairs = [
          p.get_arms()
          for p in pipelines
      ]
```

Okay, what spirals have we recovered? These are logarithmic spirals fit to our noisy data, see how close they are regardless of the spiral model used! This shows just how tricky this problem is

```
[13]  fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(7, 7.4), sharex=True,
      axes = axes.reshape(-1)
      for i, arms in enumerate(arm_pairs):
```

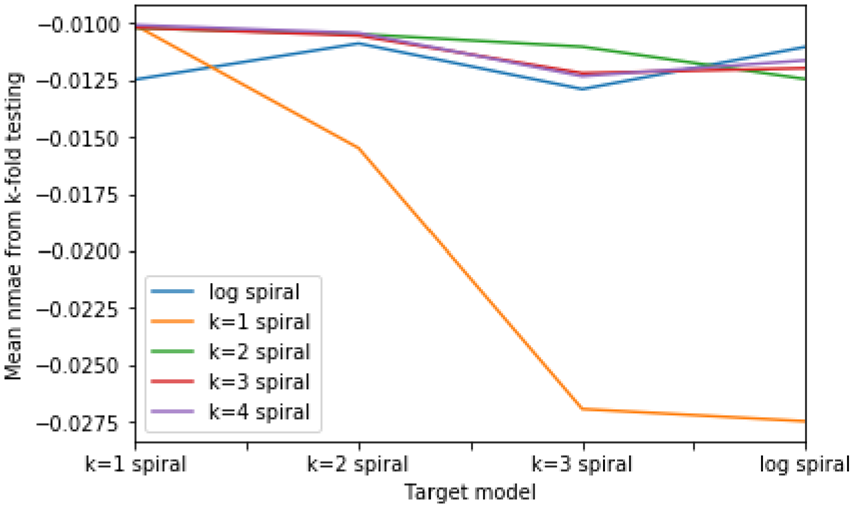
```
    for a in arms:
        axes[i].plot(*a.coords[a.outlier_mask].T, '.')
        axes[i].plot(*a.reprojected_log_spiral.T, c='r', linewidth=5, alp
    for arm in template_arms[i]:
        axes[i].plot(*arm.T, 'k--', linewidth=10, alpha=0.4)
    axes[i].set_title(labels[i])
plt.tight_layout();
```



And now we'll use group k-fold cross validation (scored using negative median absolute error) to attempt to recover the model which best describes the data.

```
[14]  df = pd.DataFrame()
      for i, arms in enumerate(arm_pairs):
          _, scores0 = arms[0].fit_polynomials()
          _, scores1 = arms[1].fit_polynomials()
          avg_scores = pd.DataFrame(scores0).append(pd.DataFrame(scores1)).mean
          df = df.append(avg_scores.transpose().rename(labels[i]))
      df.columns = ['log spiral'] + ['k={} spiral'.format(i) for i in range(1,
      df.plot()
      plt.xlabel('Target model')
      plt.ylabel('Mean nmae from k-fold testing')
      df['best model'] = df.transpose().idxmax()
      df
```

	log spiral	k=1 spiral	k=2 spiral	k=3 spiral	k=4 spiral	best model
k=1 spiral	-0.012473	-0.010093	-0.010211	-0.010193	-0.010079	k=4 spiral
k=2 spiral	-0.010887	-0.015486	-0.010477	-0.010546	-0.010429	k=4 spiral
k=3 spiral	-0.012897	-0.026931	-0.011034	-0.012195	-0.012327	k=2 spiral
log spiral	-0.011039	-0.027468	-0.012455	-0.011975	-0.011631	log spiral



Conclusions

The main message we've learnt is that this is really tricky - spirals are versatile models which are hard to distinguish between.