TP2: Face Recognition

Part 1: Harmonic Function Solution

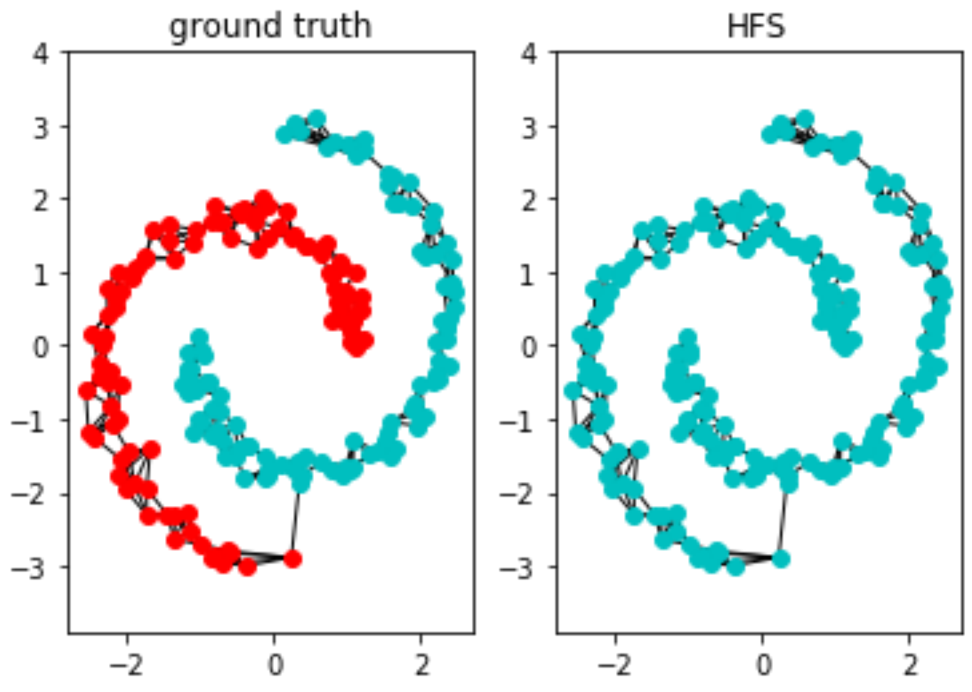
In [1]:

```
from DM2.code_material_python.harmonic_function_solution.func import*
```

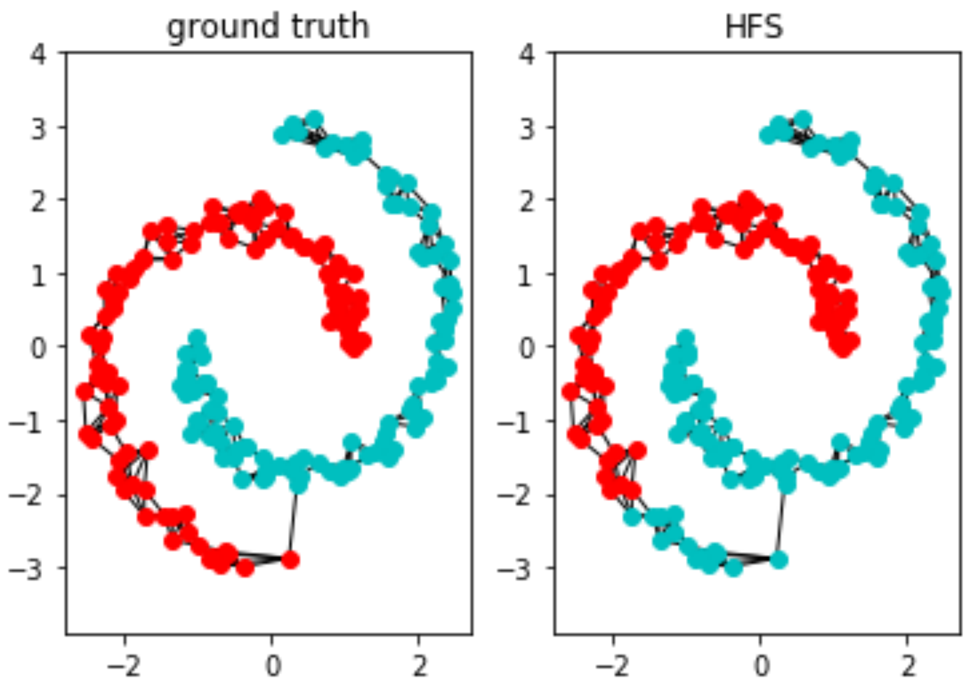
1.1. Complete hard_hfs.m and two_moons_hfs.m . Select uniformly at random 4 labels (S), and compute the labels for the unlabeled nodes (T) using the hard-HFS formula. Plot the resulting labeling and the accuracy.

In [2]:

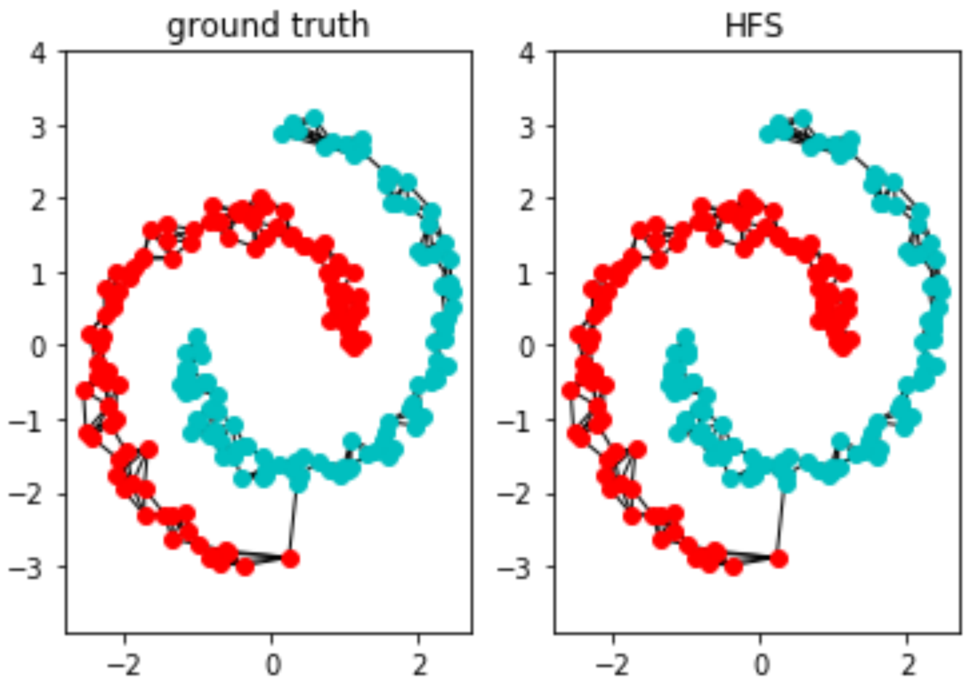
```
acc=np.zeros(5)
for i in range(5):
    acc[i] = two_moons_hfs(mislabeled=0,soft=False,laplacian_normalization="rw
",var=1,eps=0,k=5)
    print('accuracy', acc[i])
print('mean accuracy',np.mean(acc))
```



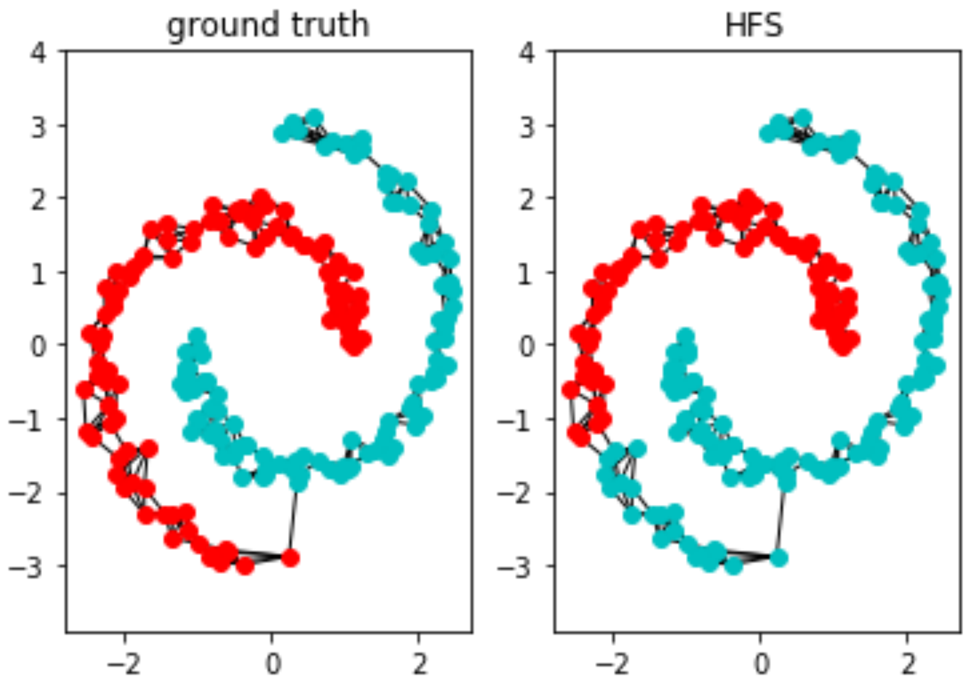
```
accuracy 0.5
```



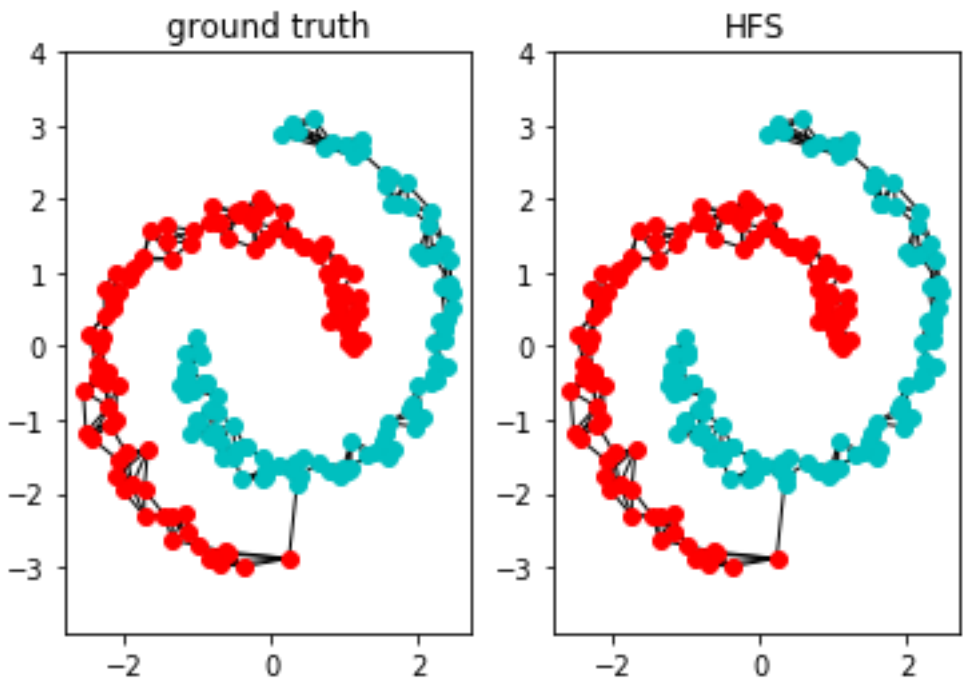
```
accuracy 0.93
```



```
accuracy 1.0
```



```
accuracy 0.895
```

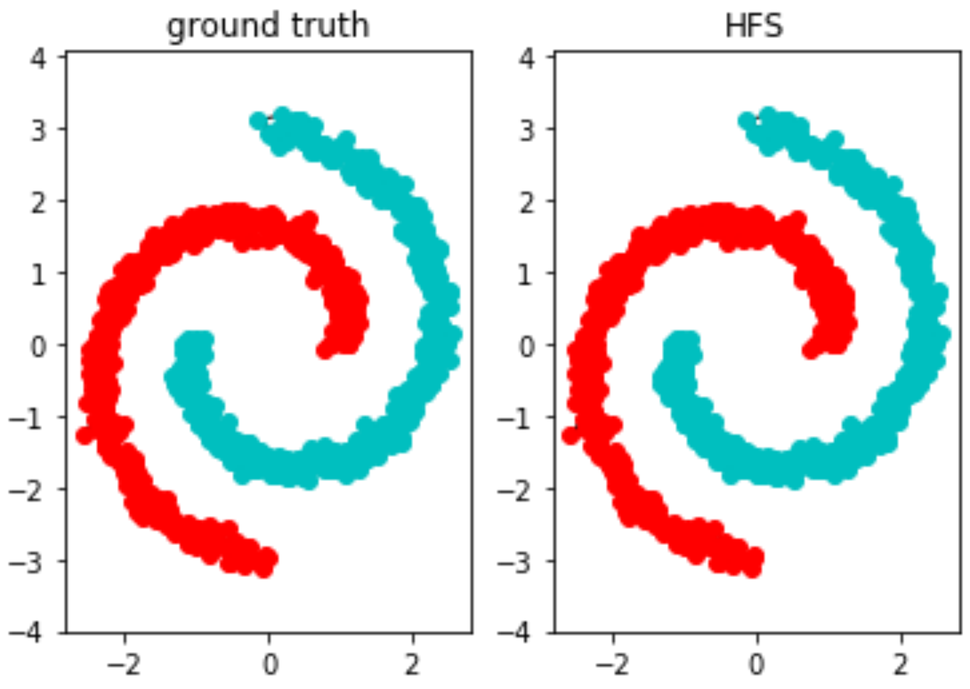


```
accuracy 1.0
mean accuracy 0.865
```

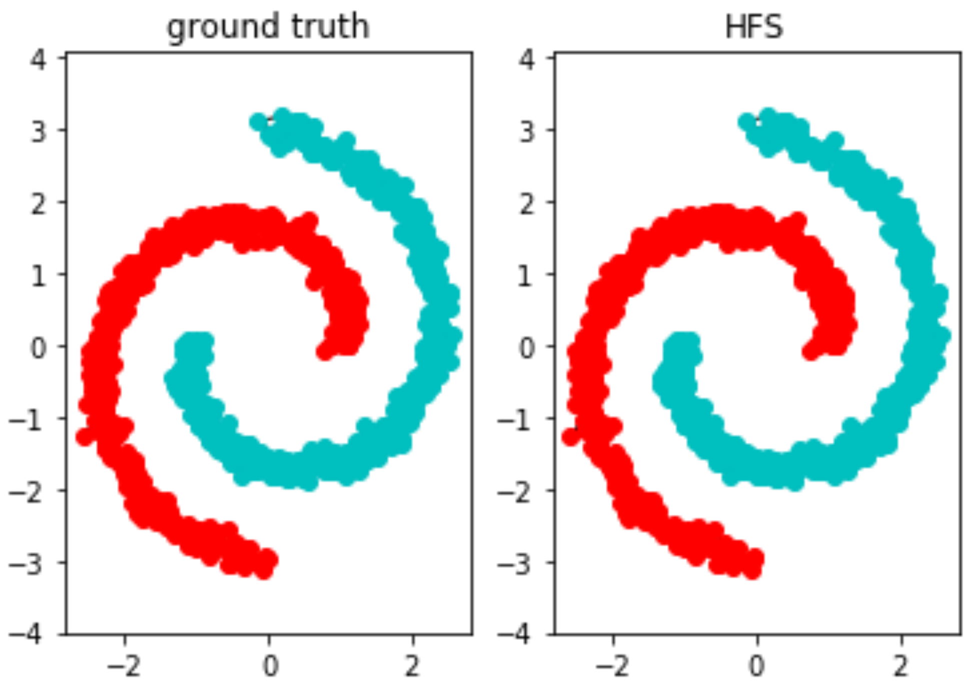
At home, run two_moons_hfs.m using the data_2moons_large. mat, a dataset with 1000 samples. Continue to uniformly sample only 4 labels. What can go wrong? The samples could all belong to the same label.

In [3]:

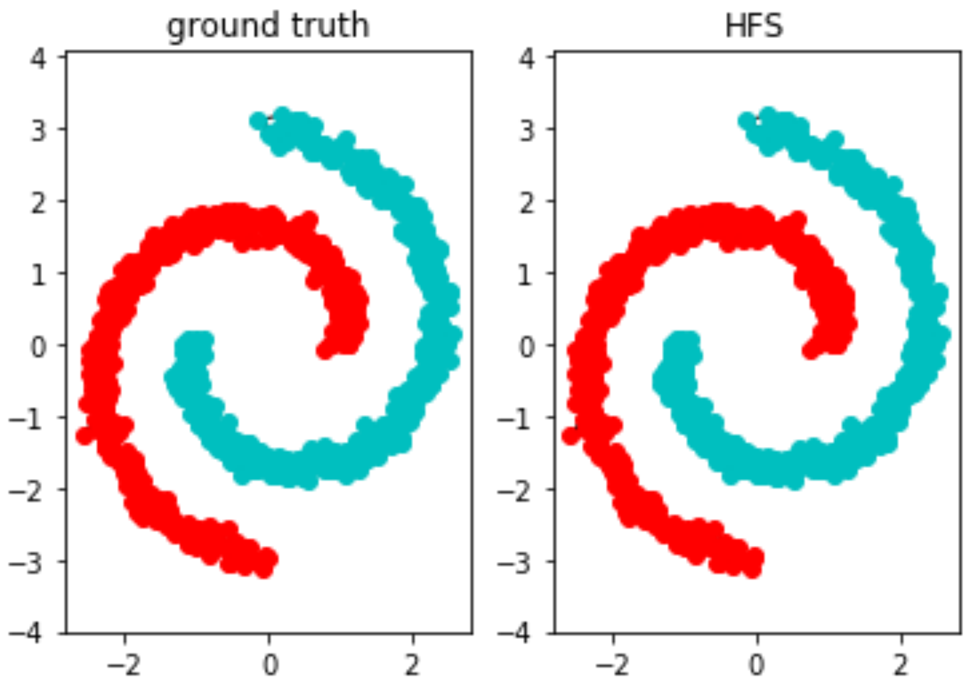
```
acc=np.zeros(5)
for i in range(5):
    acc[i] = two_moons_hfs(mislabeled=0,soft=False,laplacian_normalization="rw
",var=1,eps=0,k=5,large=1)
    print(acc[i])
print('mean accuracy',np.mean(acc))
```



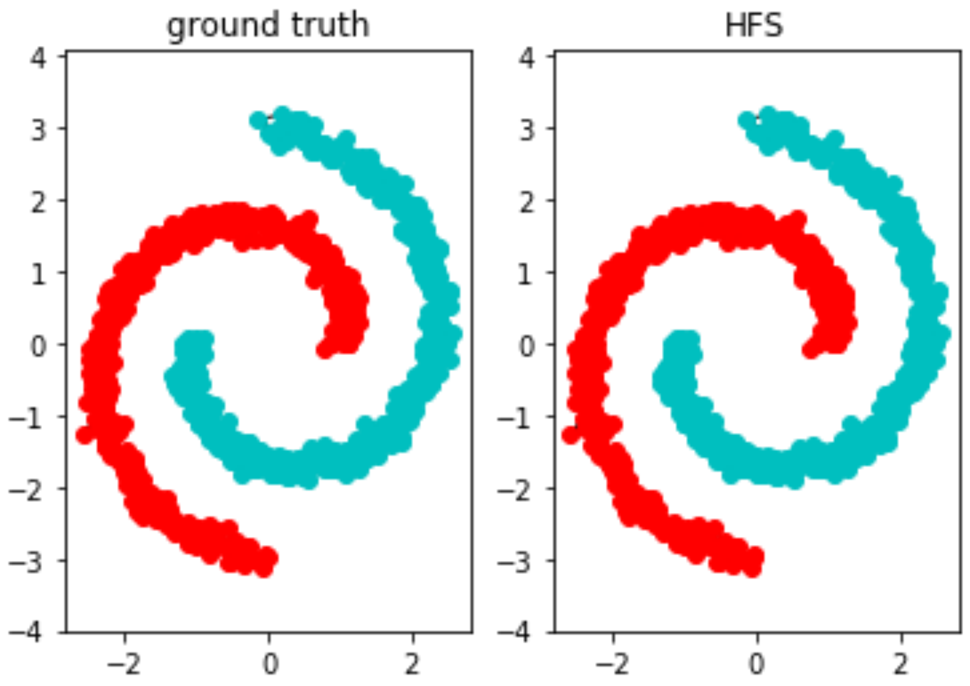
```
1.0
```



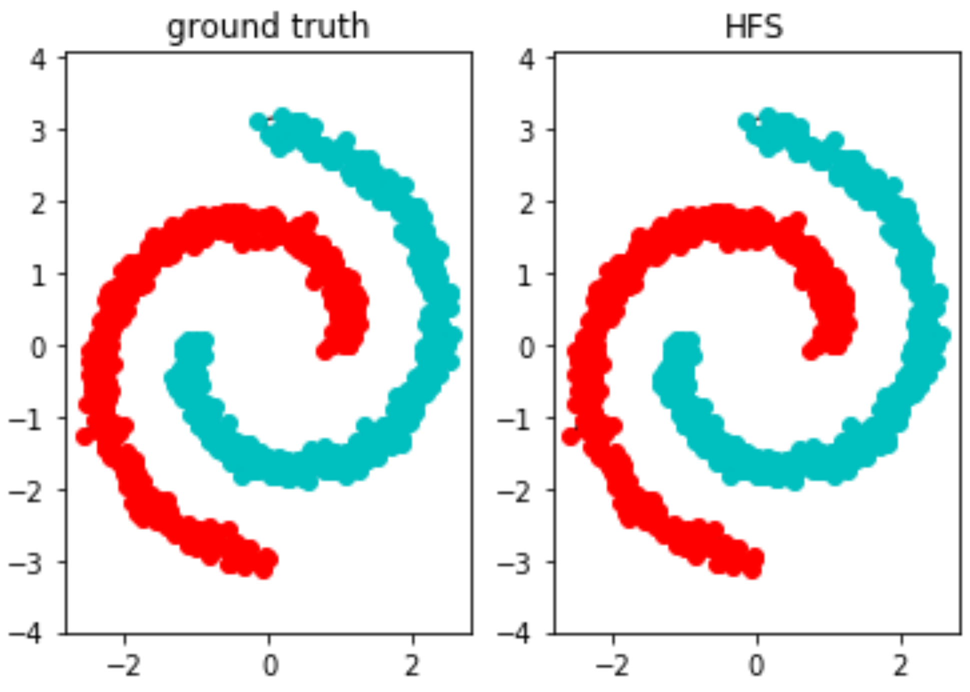
```
1.0
```



```
1.0
```



```
1.0
```

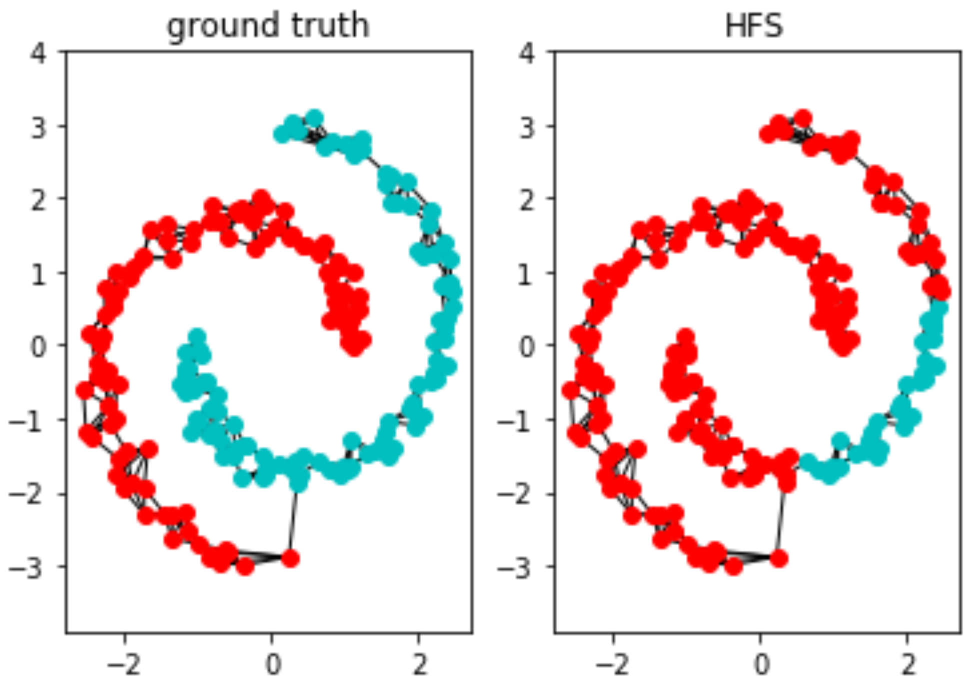


```
1.0
mean accuracy 1.0
```

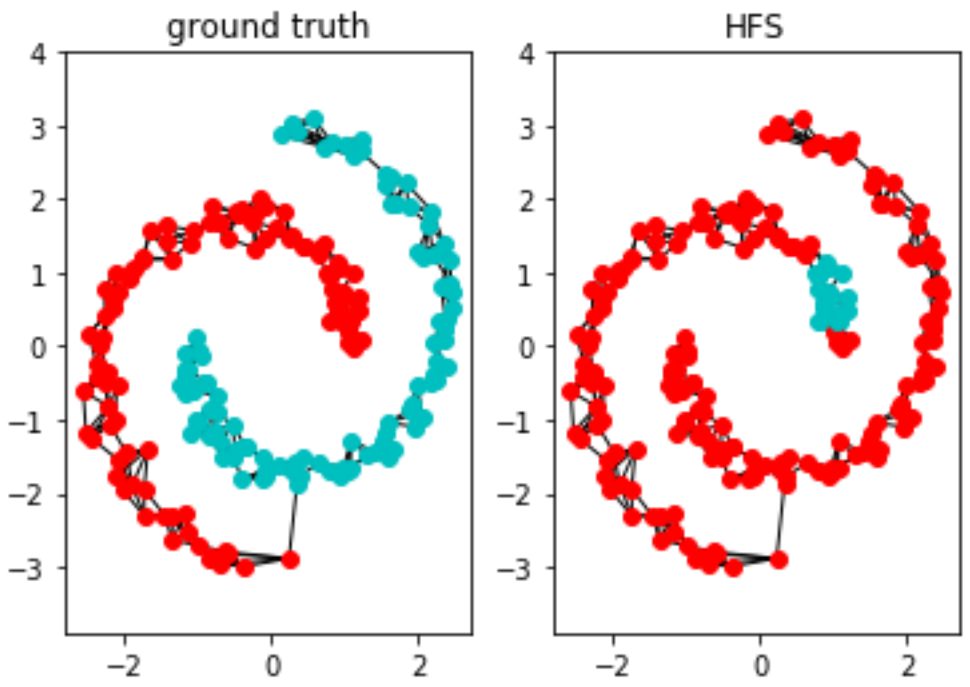
What happens when the labels are noisy, or in other words when some of the samples are mislabeled? Hard harmonic functions performs badly when the samples are mislabeld unlike soft harmonic function.

In [4]:

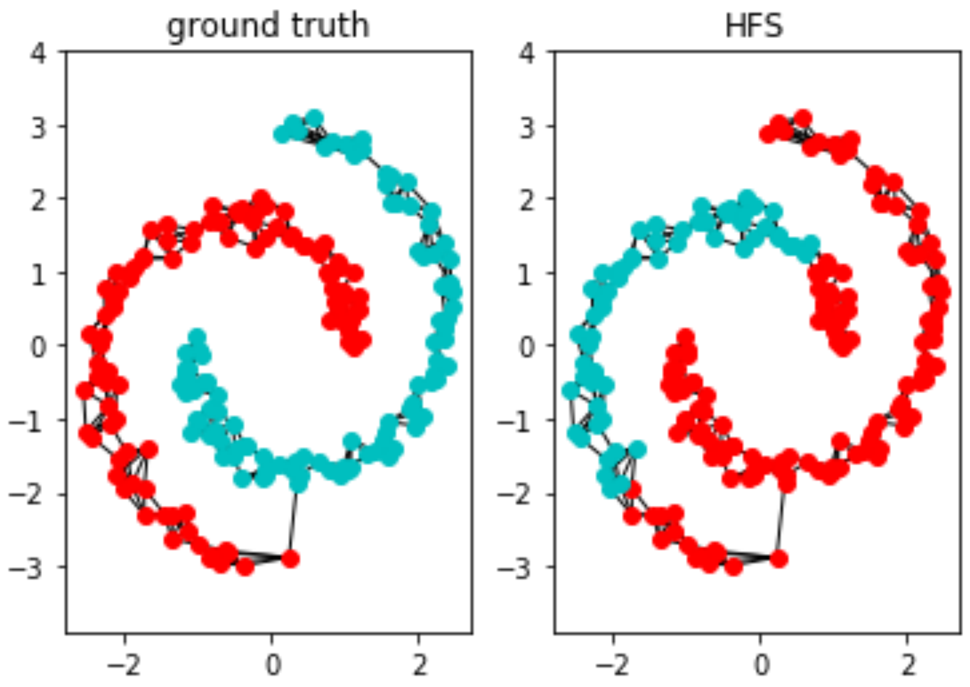
```
acc=np.zeros(5)
for i in range(5):
    acc[i] = two_moons_hfs(mislabeled=1,soft=False,laplacian_normalization="rw
",var=1,eps=0,k=5)
    print(acc[i])
print('mean accuracy',np.mean(acc))
```



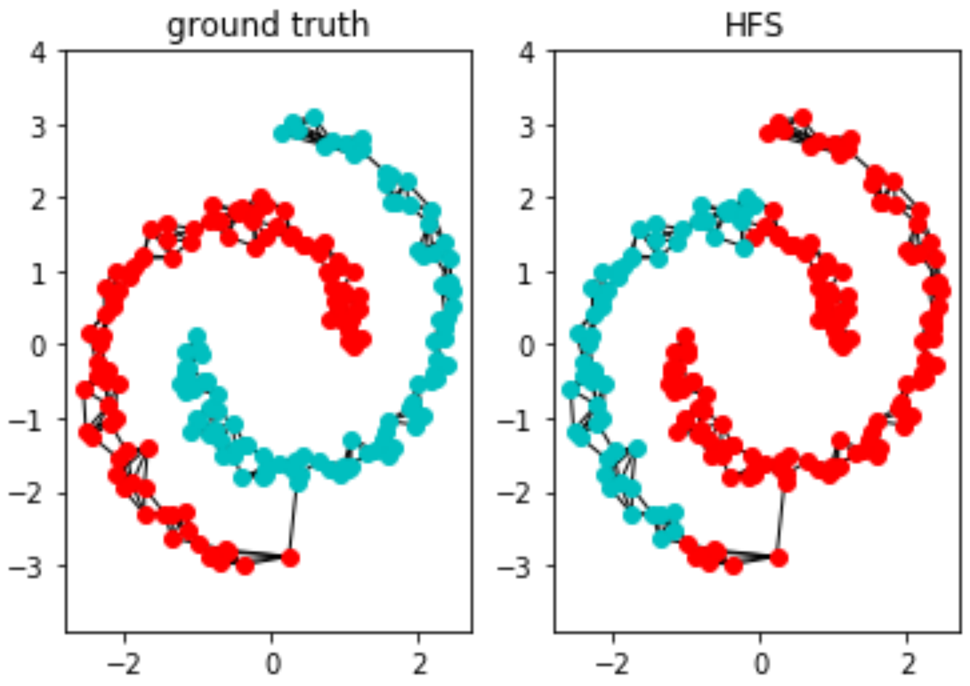
```
0.66
```



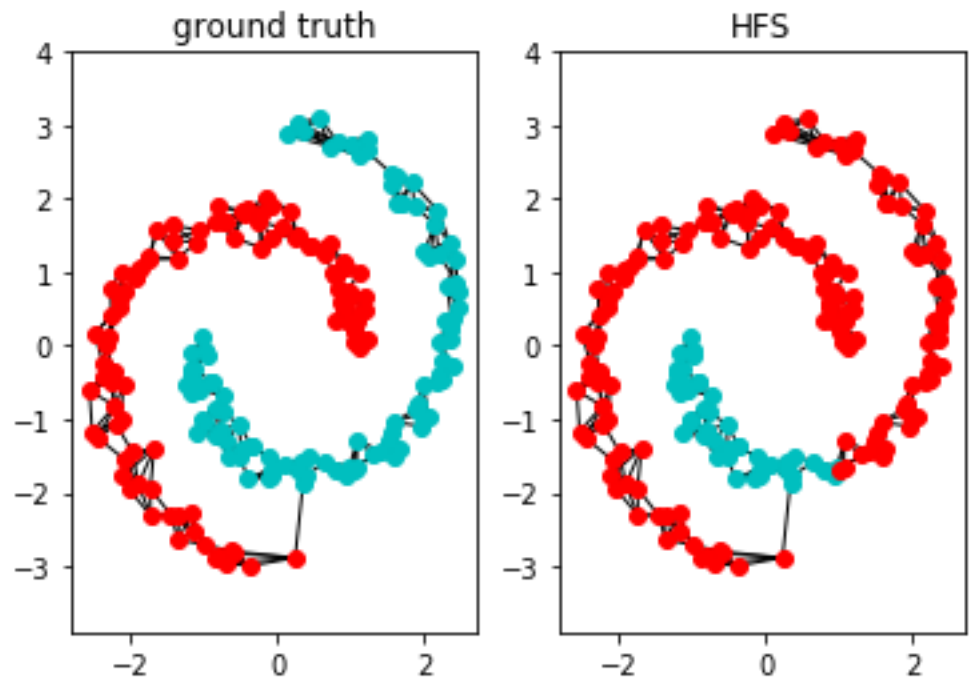
```
0.41
```



```
0.19
```



```
0.205
```

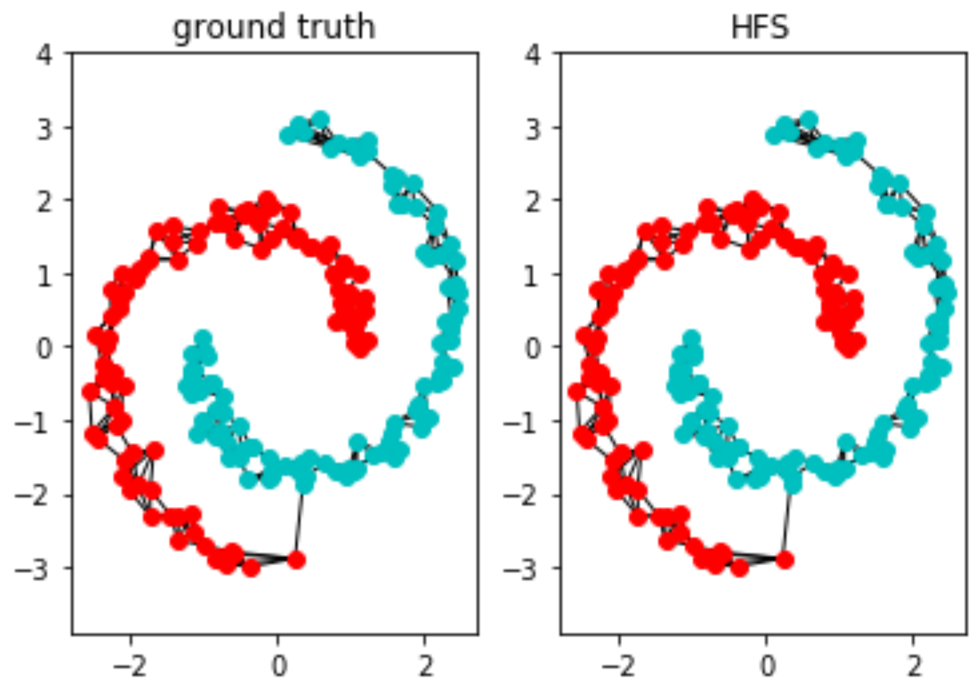


```
0.725
mean accuracy 0.438
```

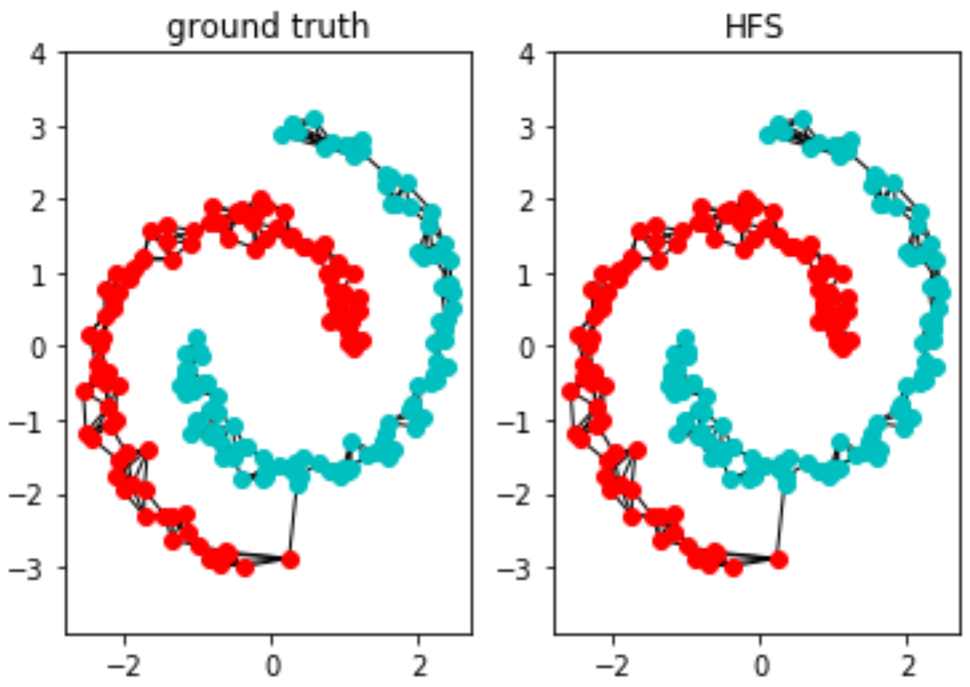
Complete soft_hfs.m and test it with two_moons_hfs.m . Now complete hard_vs_soft_hfs.m.

In [12]:

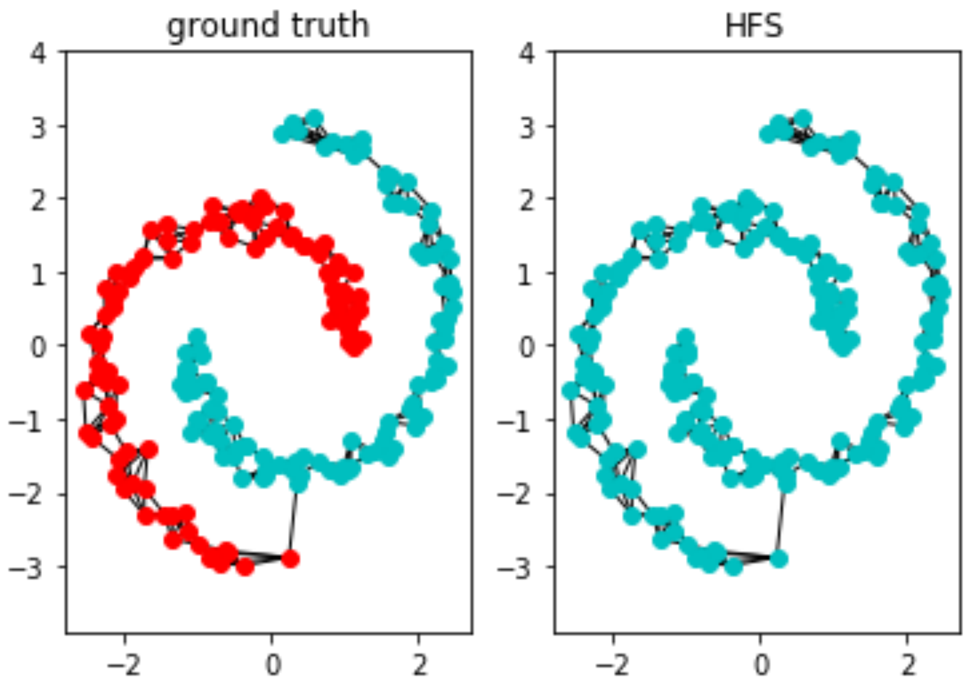
```
acc=np.zeros(5)
for i in range(5):
    acc[i] = two_moons_hfs(mislabeled=0,soft=True,laplacian_normalization='rw'
,var=1,eps=0,k=5)
    print(acc[i])
print('mean accuracy',np.mean(acc))
```



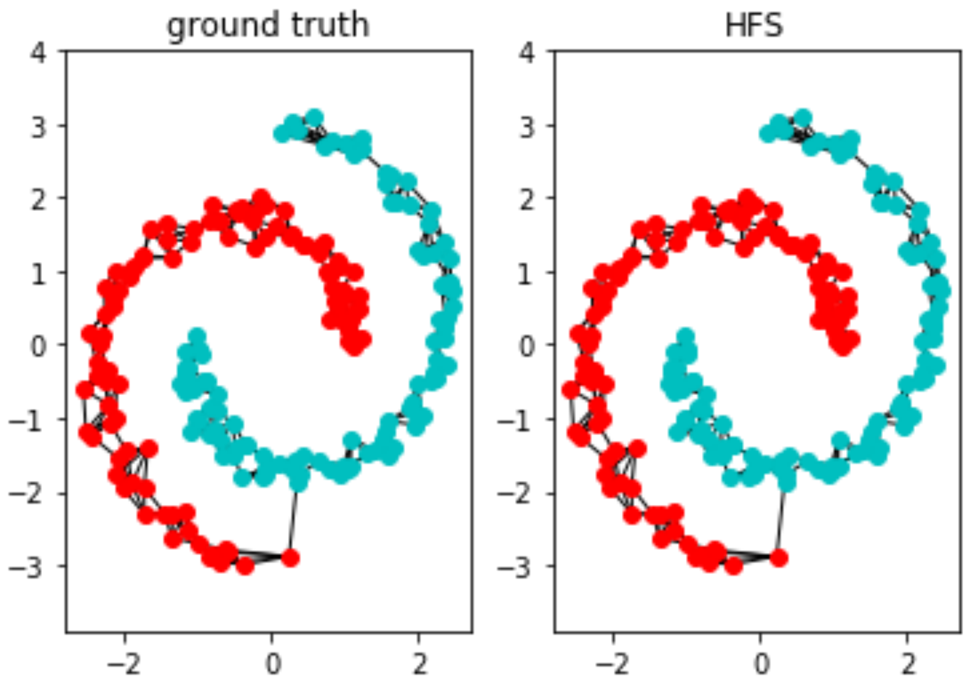
```
1.0
```



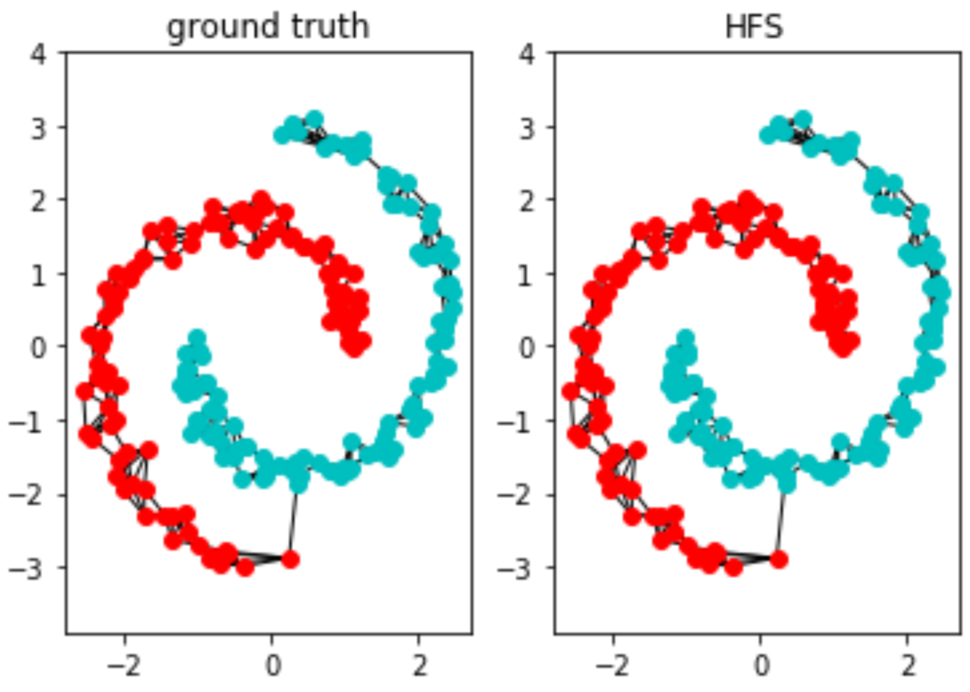
```
1.0
```



```
0.5
```



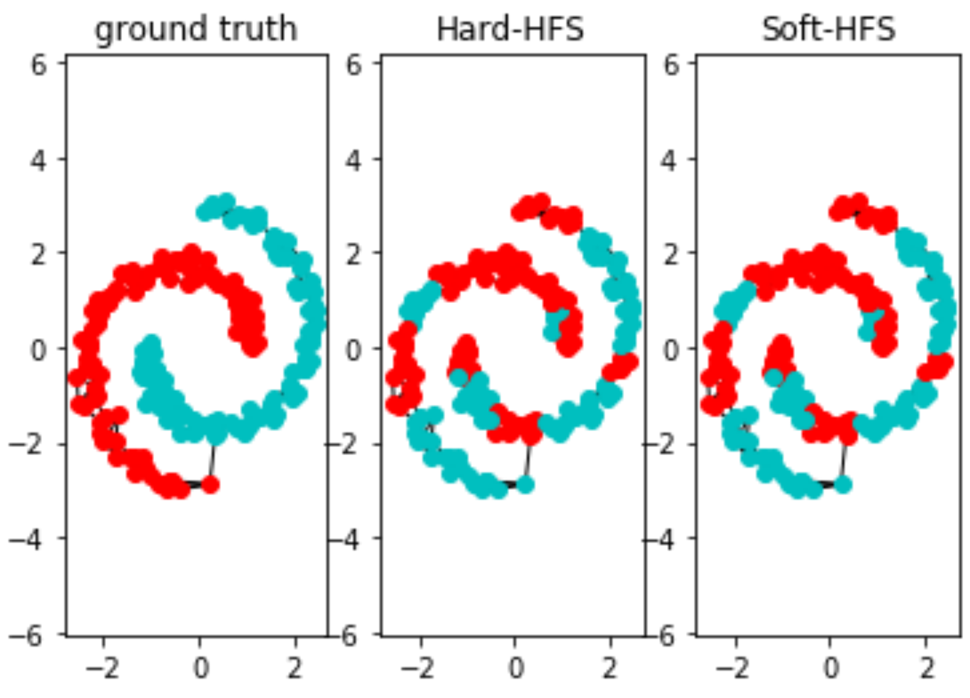
```
1.0
```



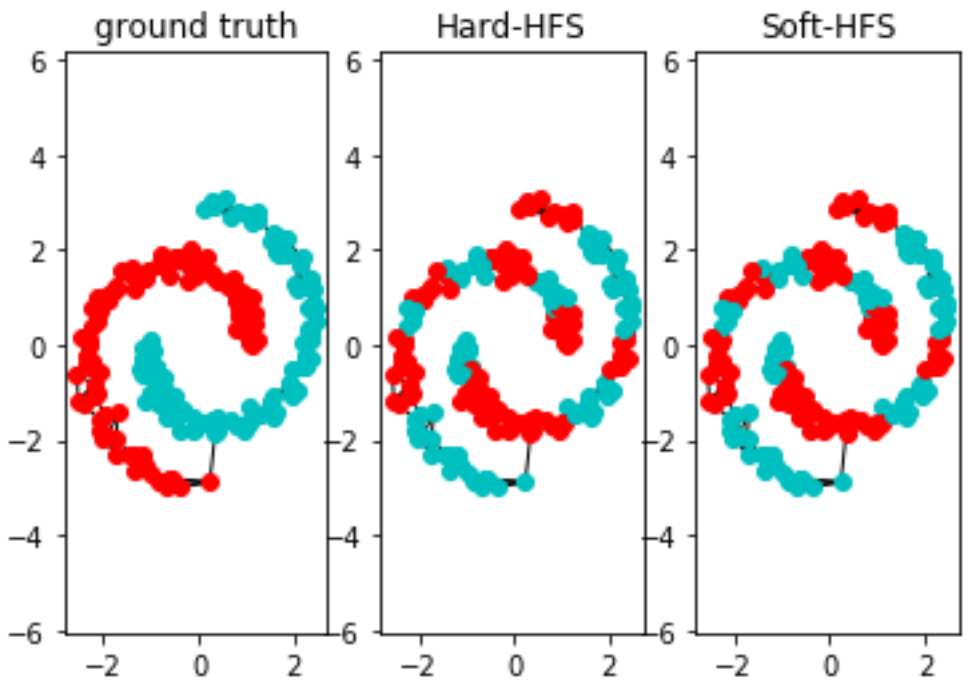
```
1.0
mean accuracy 0.9
```

In [5]:

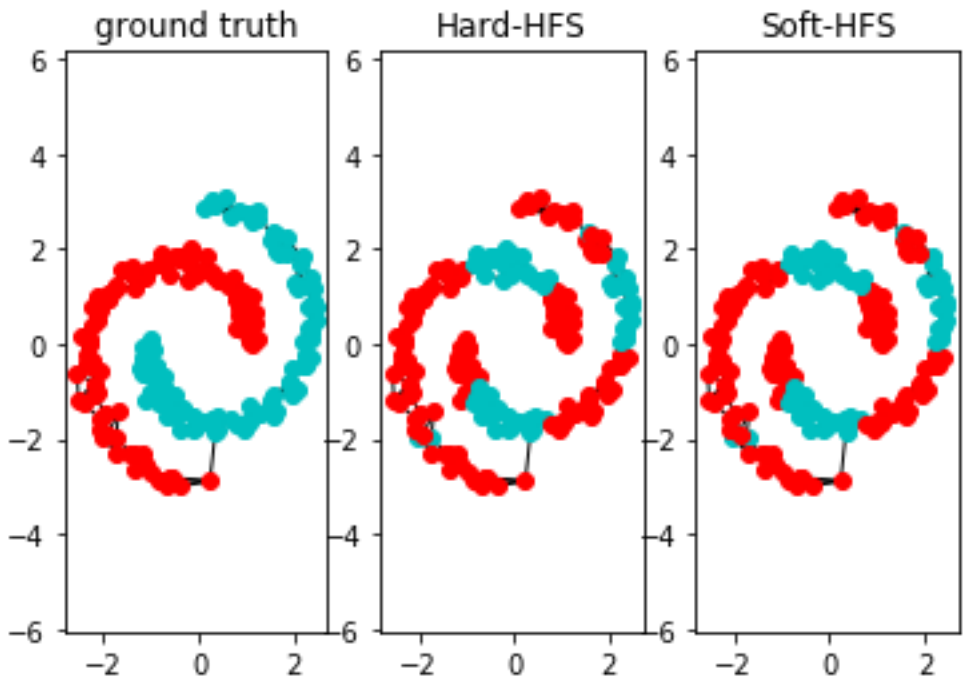
```python
acc=[]
for i in range(5):
    acc.append(hard_vs_soft_hfs(laplacian_normalization='rw',var=1,eps=0,k=5))
    print(acc)
print('mean accuracy',np.mean(acc,axis=0))
```



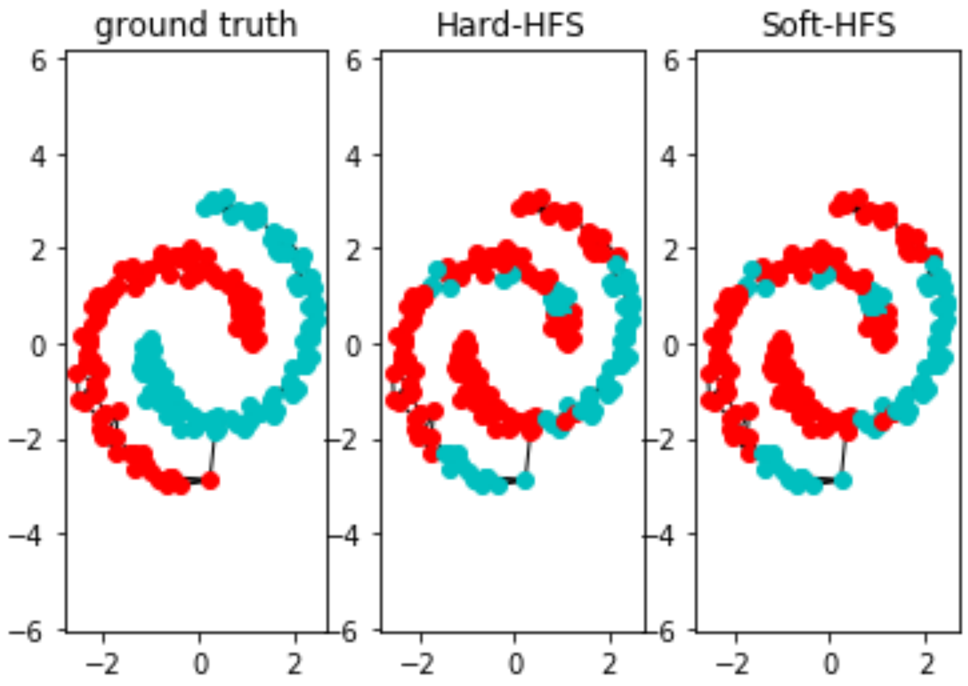
```
[[0.6, 0.5]]
```



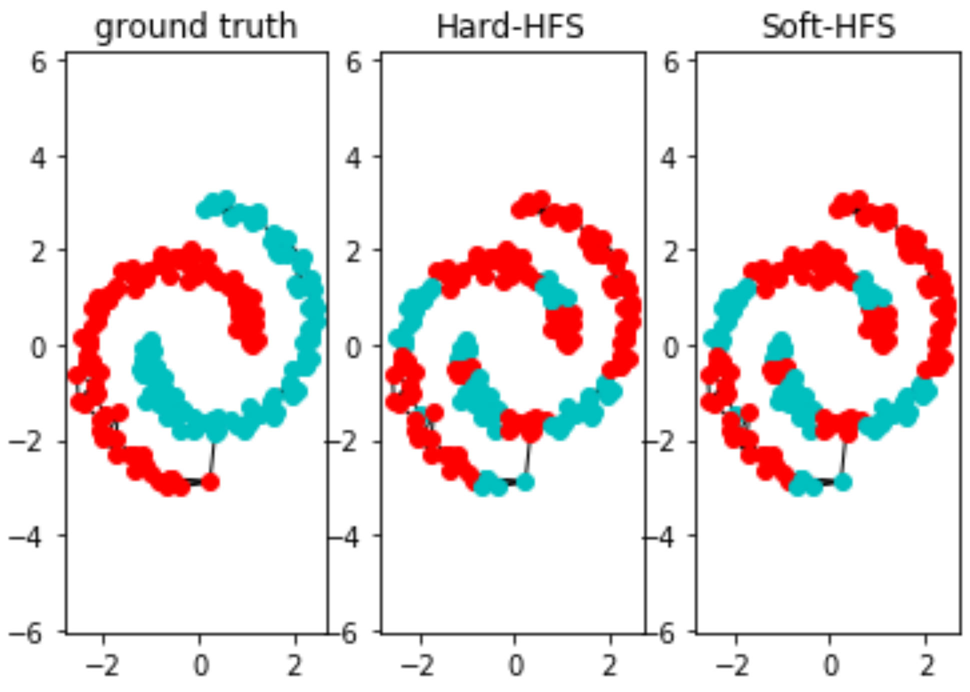
```
[[0.6, 0.5], [0.5, 0.5]]
```



```
[[0.6, 0.5], [0.5, 0.5], [0.58, 0.5]]
```



```
[[0.6, 0.5], [0.5, 0.5], [0.58, 0.5], [0.555, 0.5]]
```



```
[[0.6, 0.5], [0.5, 0.5], [0.58, 0.5], [0.555, 0.5], [0.55, 0.5]]
mean accuracy [0.557 0.5   ]
```

Compare the results you obtain with soft-HFS anc hard-HFS.

Weirdly Hard-HFS performs slightly better in presence of noisy data, I'm not sure this is supposed to happen..

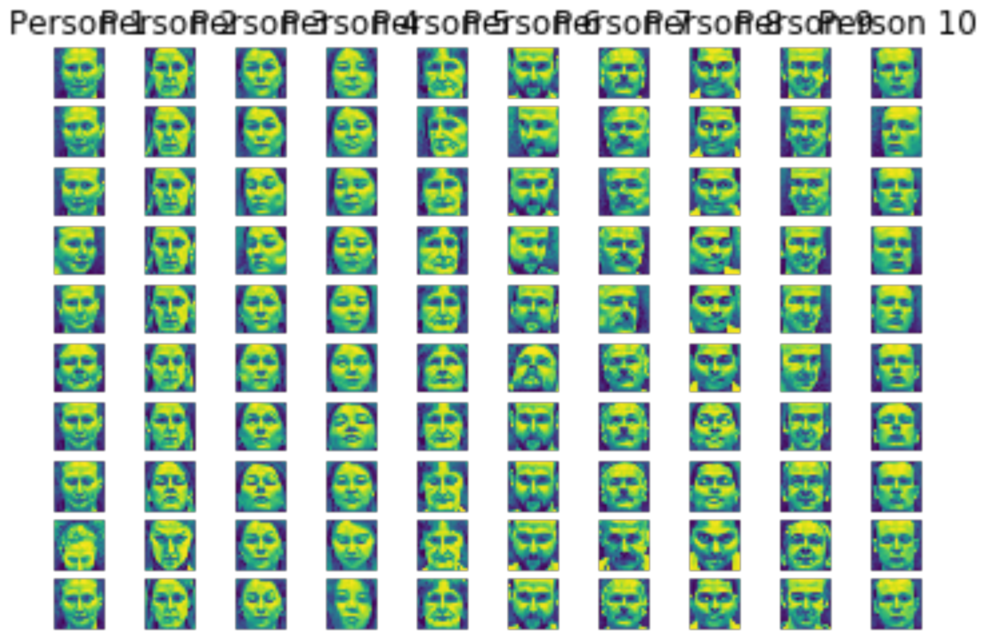
Part 2: Face recognition with HFS

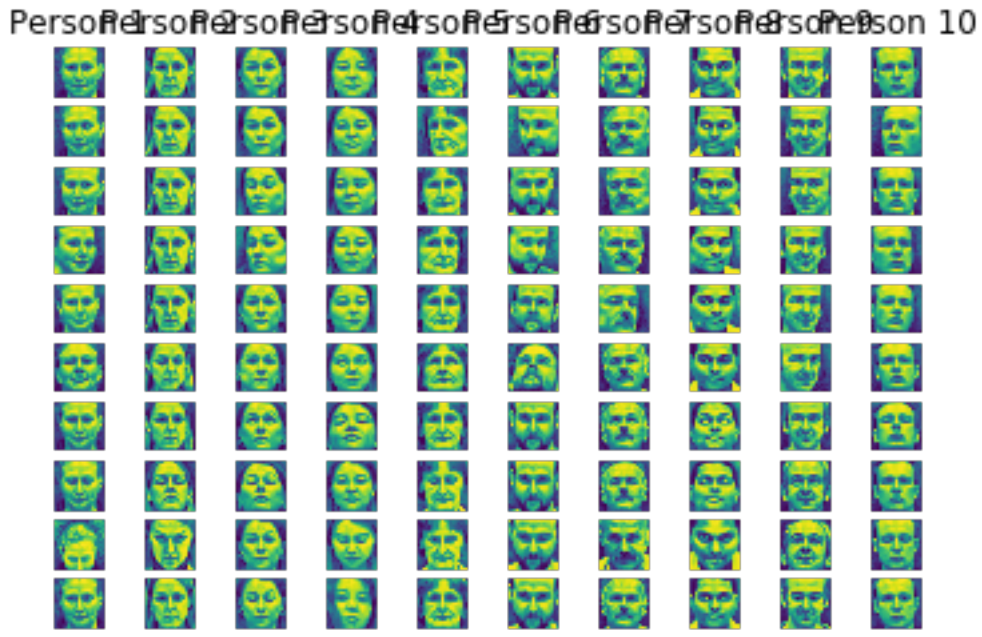
In [2]:

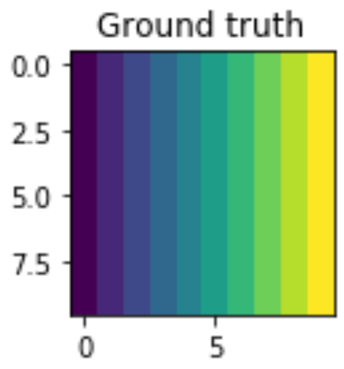
```
from DM2.code_material_python.Face_recognition_with_HFS.func import*
```

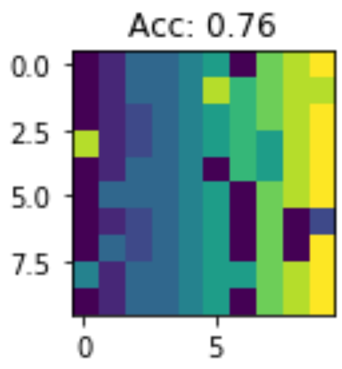
In [5]:

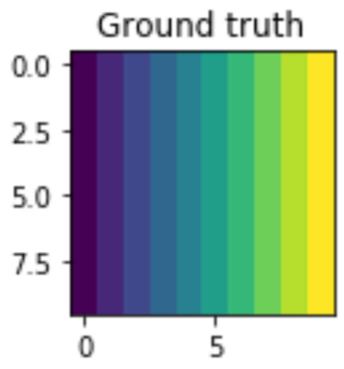
```
offline_face_recognition(laplacian_normalization='rw')
```

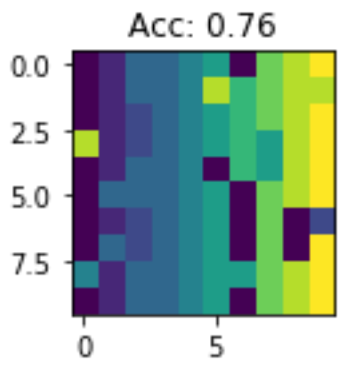

Person 1
Person 2
Person 3
Person 4
Person 5
Person 6
Person 7
Person 8
Person 9
Person 10


Person 1
Person 2
Person 3
Person 4
Person 5
Person 6
Person 7
Person 8
Person 9
Person 10


Ground truth
0.0
2.5
5.0
7.5
0
5


Acc: 0.76
0.0
2.5
5.0
7.5
0
5


Ground truth
0.0
2.5
5.0
7.5
0
5


Acc: 0.76
0.0
2.5
5.0
7.5
0
5

2.1. How did you manage to label more than two classes?

I used hot encoding in the soft/hard harmonic functions

2.2. Which preprocessing steps (e.g. cv.GaussianBlur, cv.equalizeHist) did you apply to the faces before constructing the similarity graph? Which gave the best performance?

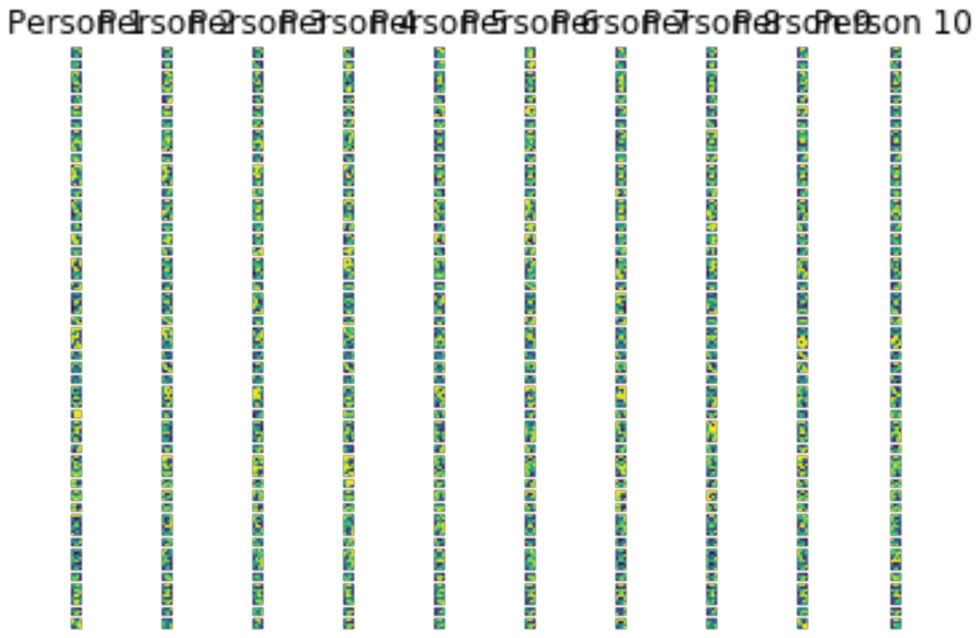
I used cv.GaussianBlur then cv.equalizeHist but most of the information in the face area was lost because its histogram is not confined to a particular region To solve this problem, I used adaptive histogram equalization that divides the image into small blocks and then applies histogram equalized.

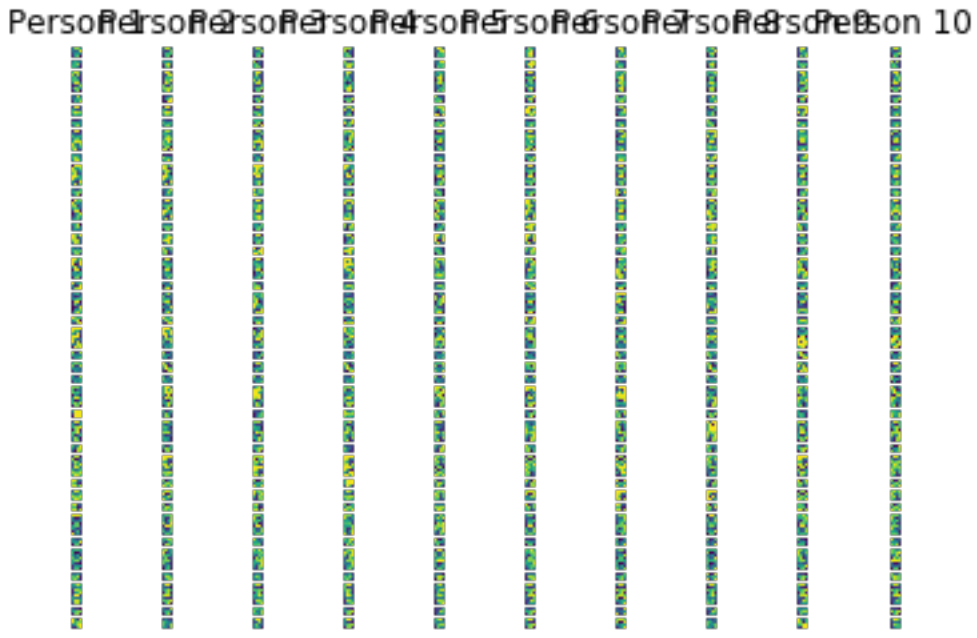
2.3. Does HFS reach good performances on this task?

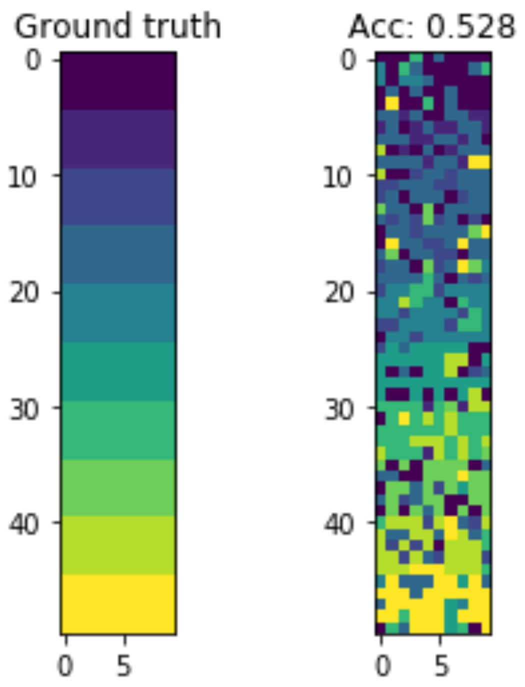
Yes!

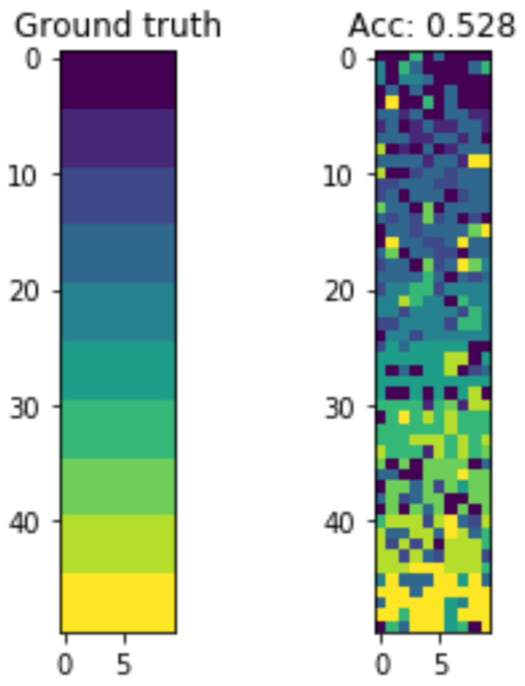
In [6]:

```
offline_face_recognition_augmented(laplacian_normalization='rw')
```









2.4. Did adding more data to the task improve performance? If so, which kind of additional data improves performance.

In the second function we use a bigger dataset probably obtained using data augmentation: Face images with different poses and illuminations. However it doesn't improve the performance.

2.5. If the performance does not improve when adding additional data, try to justify why. Which kind of additional data degrades performance instead of improving it?

First of all we only show 4 labeled data in both cases (but the first one has 10 images and the second one has 50) So based only on the number of showed labels we can attain an accuracy of 40% in the first one against 8% in the second case. But the soft harmonic function is supposed to perform well even with few labels. So I think that the problem comes from the fact that the augmented dataset had a lot of fuzzy images (probably lot of pictures have been taking while the person was moving her face) which is why it degrades performance instead of improving it.