Decoded_Orientation

July 15, 2022

1 Decoding population activity

Imports

[ ]:
```
%matplotlib inline
from sym_model import Population, sigmoid

import sys
import seaborn as sns
import umap
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
sys.path.insert(0, './model')

from model import decoding
from model import persistence
from utils import get_orientation_phase_grid
```

1.1 Getting the data

Sampling the activity using the model

[ ]:
```
limit = 500
# n_theta, n_phi = 18, 24
n_theta, n_phi = 36, 72
# n_theta, n_phi = 9, 12
step_phi, step_theta = 360 // n_phi, 180 // n_theta
N = 40 # number of cells
```

[ ]:
```
grid = get_orientation_phase_grid(step_phi, step_theta)
grid = grid.reshape((-1, 2))
print(grid[0], grid[-1], grid.shape)
```

```
[0. 0.] [3.05432619 6.19591884] (2592, 2)
```

[ ]:
```
population = Population.random(N)
```

[ ]:
```
phi_deg, theta_deg = grid[:, 1] * 180 / np.pi, grid[:, 0] * 180 / np.pi
```

[ ]:
```
# res = population.sample_responses(limit, custom_grid=grid, use_sigmoid=False)
# res, phi_deg, theta_deg = res[:, :, 0], res[:, 0, 1], res[:, 0, 2]
```

[ ]:
```
res = np.abs(population.response_func(grid[:, 1], grid[:, 0]).swapaxes(0, 1)) *␣
↪10
```

[ ]:
```
phi_reorder = sorted(list(range(len(phi_deg[:limit]))), key=lambda x:␣
↪phi_deg[x])
theta_reorder = sorted(list(range(len(theta_deg[:limit]))), key=lambda x:␣
↪theta_deg[x])
```

[ ]:
```
res.shape
```

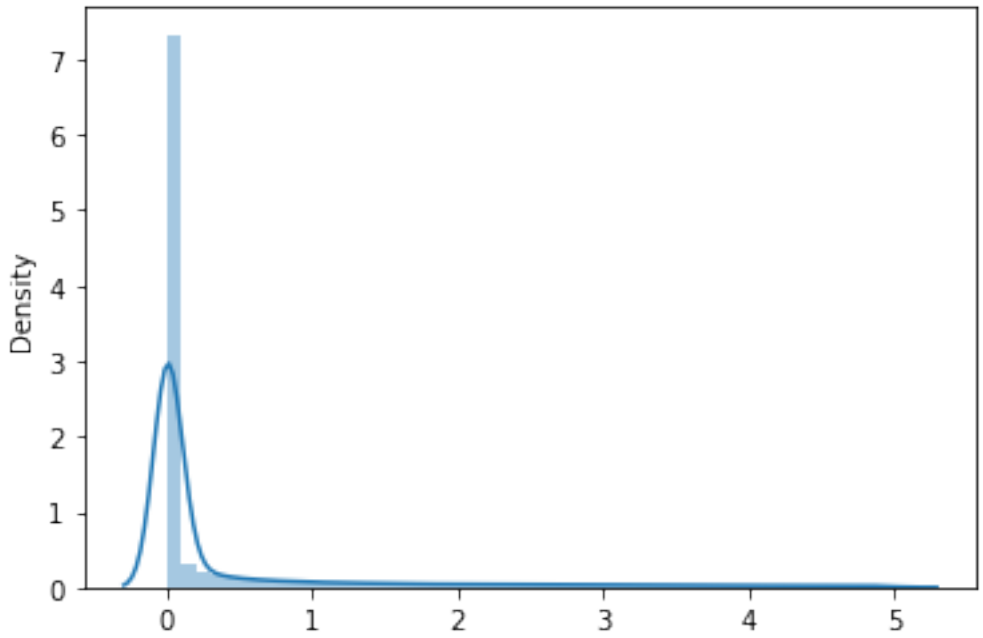
[ ]:
$(2592,\ 40)$

[ ]:
```
res_reshaped = res.reshape((n_phi, n_theta, -1))
```

[ ]:
```
sns.distplot(res.ravel())
```

```
/usr/lib/python3.10/site-packages/seaborn/distributions.py:2619: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version.
Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

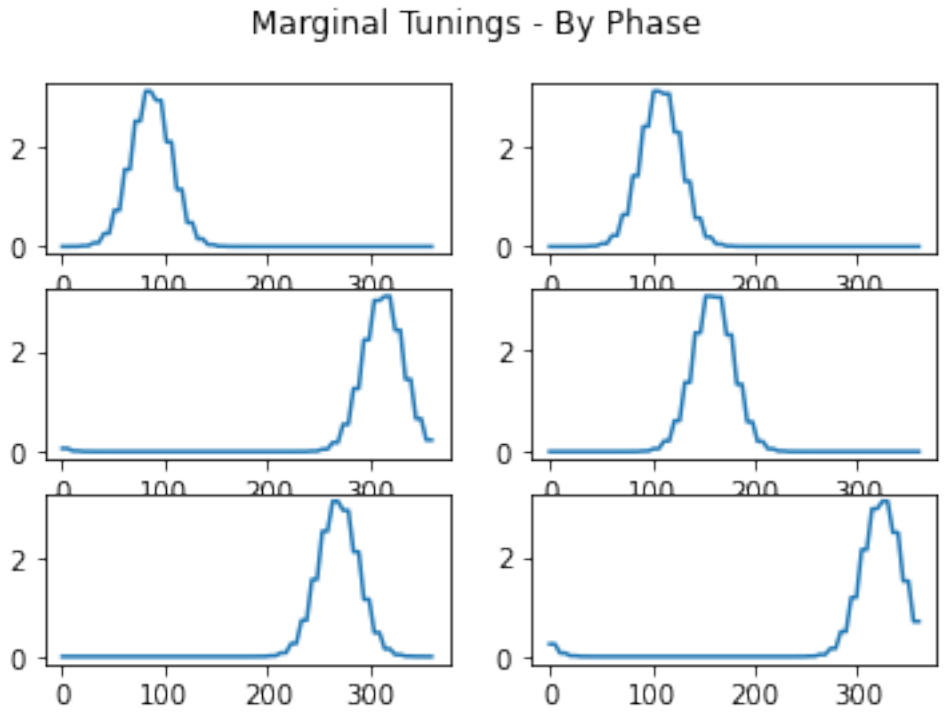
[ ]:
```
<AxesSubplot:ylabel='Density'>
```



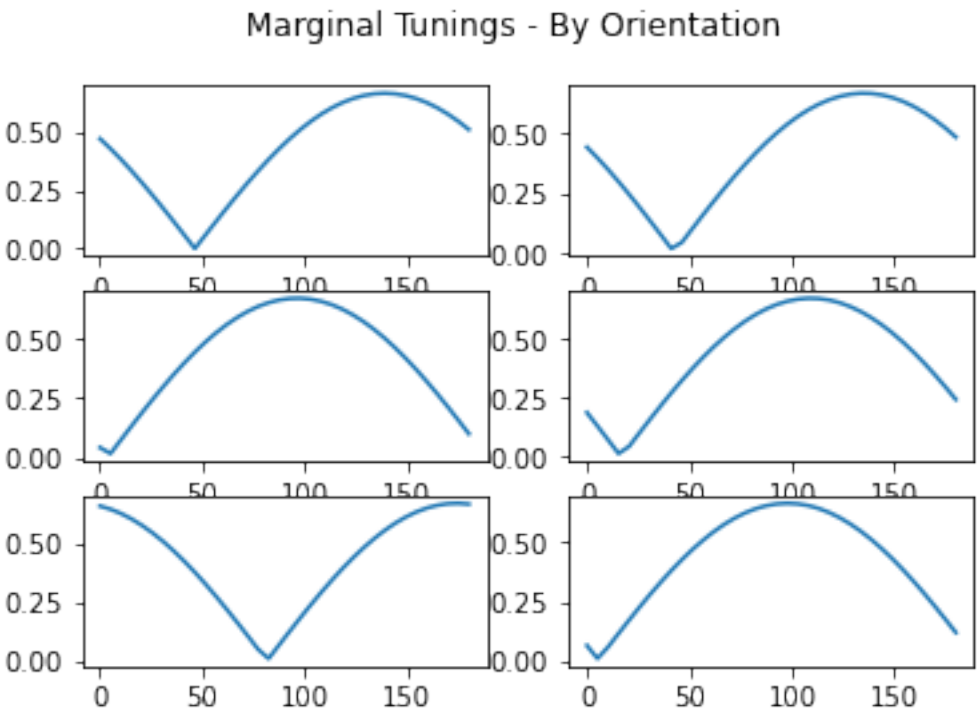
1.2 Plotting joint tunings

[ ]:
```
phase_linspace = np.linspace(0, 360, n_phi)
orientation_linspace = np.linspace(0, 180, n_theta)
```

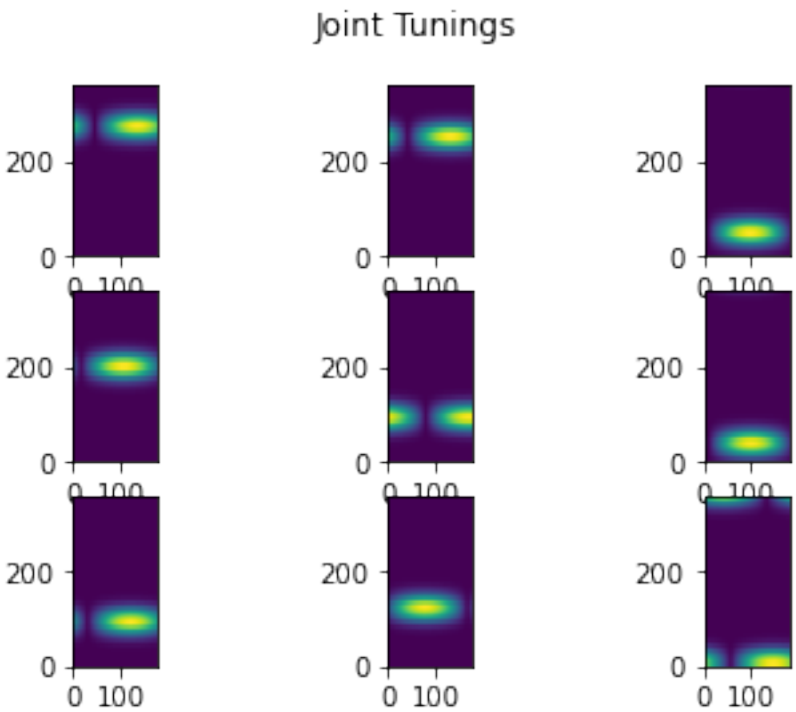
[ ]:
```
fig, ax = plt.subplots(3, 2)
ax = ax.flatten()
# set title
fig.suptitle('Marginal Tunings - By Phase')
for i in range(6):
    ax[i].plot(phase_linspace, res_reshaped.mean(1)[:, i])
```



[ ]:
```
fig, ax = plt.subplots(3, 2)
ax = ax.flatten()
# set title
fig.suptitle('Marginal Tunings - By Orientation')
for i in range(6):
    ax[i].plot(orientation_linspace, res_reshaped.mean(0)[:, i])
```

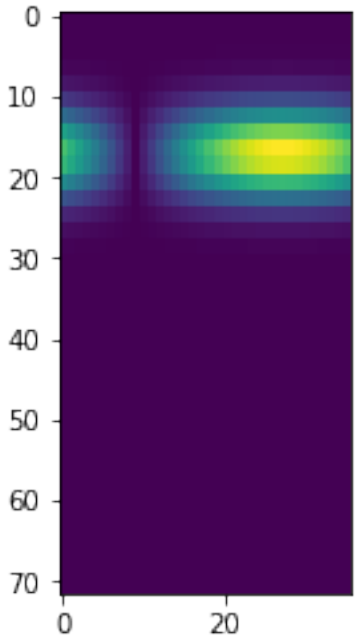


[ ]:
```
fig, ax = plt.subplots(3, 3)
ax = ax.flatten()
# set title
fig.suptitle('Joint Tunings')
for i in range(9):
    ax[i].imshow(res_reshaped[:, :, i], cmap='viridis', extent=[0, 180, 0, 360])
```



[ ]:
```
plt.imshow(res_reshaped[:, :, 0], cmap='viridis')
```

[ ]:
```
<matplotlib.image.AxesImage at 0x7fefe9397f40>
```



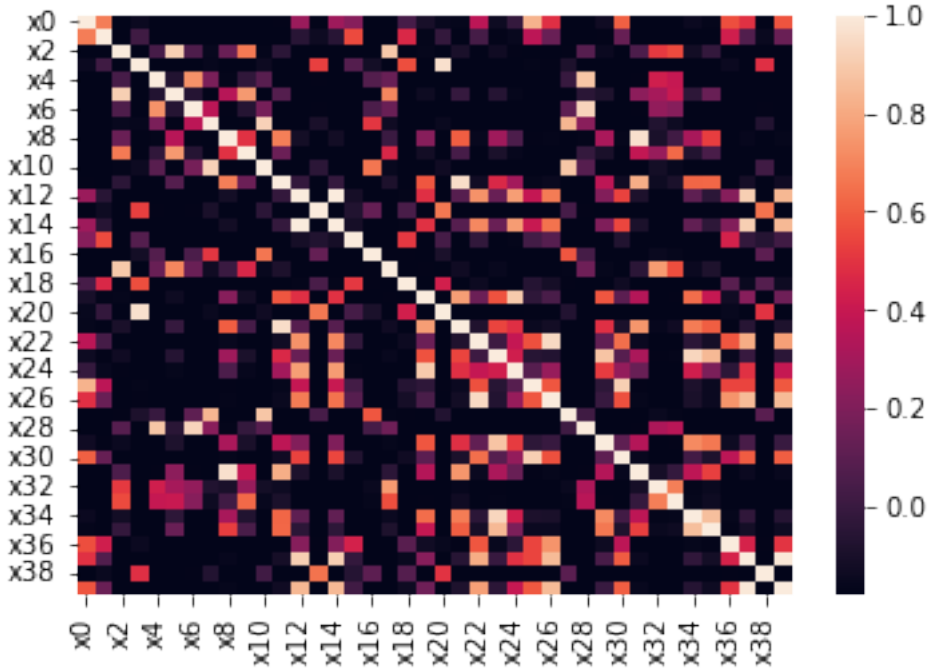
1.3 PCA

[ ]:
```
res_df = pd.DataFrame(res, columns=['x' + str(i) for i in range(N)])
```

[ ]:
```
from sklearn.decomposition import PCA
import seaborn as sns
```

[ ]:
```
sns.heatmap(res_df.corr())
```

[ ]: <AxesSubplot:>



Let's take 6 components

[ ]:
```
N_COMP = 4
pca = PCA(n_components=N_COMP)
pca.fit(res_df)
data_pca = pca.transform(res_df)
data_pca = pd.DataFrame(data_pca, columns=[f'PC{i+1}' for i in range(N_COMP)])
data_pca.head()
```

[ ]:

	PC1	PC2	PC3	PC4
0	1.152136	6.882894	-1.545937	-0.027478
1	1.081195	6.691970	-1.446043	-0.080447

```
2  0.996579  6.447147 -1.338007 -0.133982
3  0.937118  6.268196 -1.259540 -0.170500
4  0.938863  6.362411 -1.261050 -0.188440
```

[ ]:
```
data_pca.shape
```

[ ]:
(2592, 4)

1.4 UMAP decoding

[ ]:
```
# reducer = umap.UMAP()
# embedding = reducer.fit_transform(data_pca.to_numpy())
```

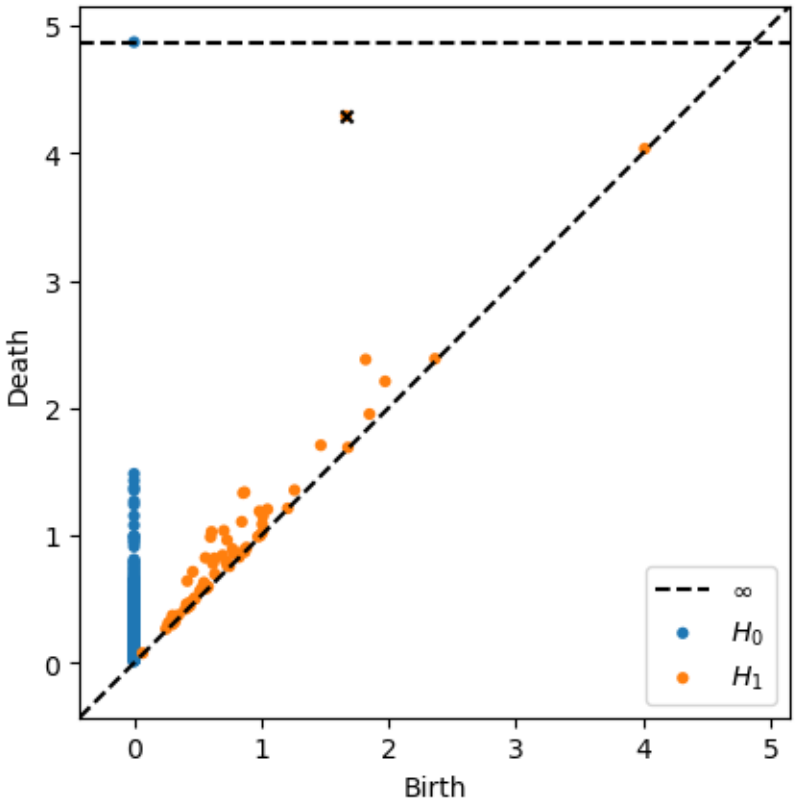
[ ]:
```
# %matplotlib notebook
# plt.scatter(
#     embedding[:, 0],
#     embedding[:, 1])
# plt.show()
```

[ ]:
```
# embedding.shape
```

1.5 Cohomological decoding

[ ]:
```
step = 5
```

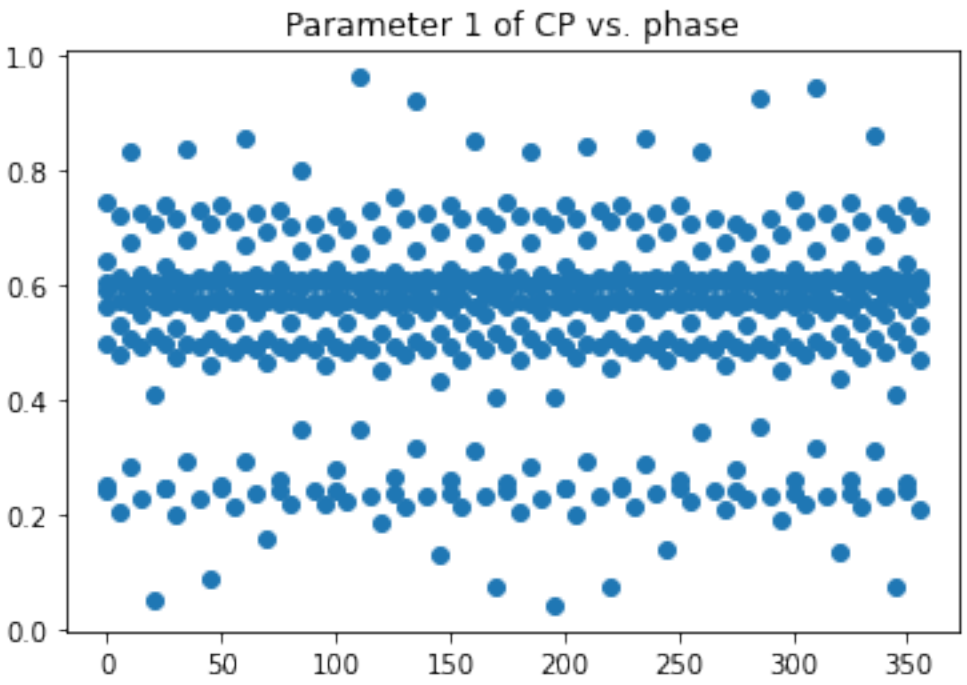
[ ]:
```
param_1 = decoding.cohomological_parameterization(data_pca[::step]).to_numpy()
```




```
Decoding… done
```

[ ]:
```
%matplotlib inline
plt.title("Parameter 1 of CP vs. phase")
plt.scatter(phi_deg[::step], param_1)
```

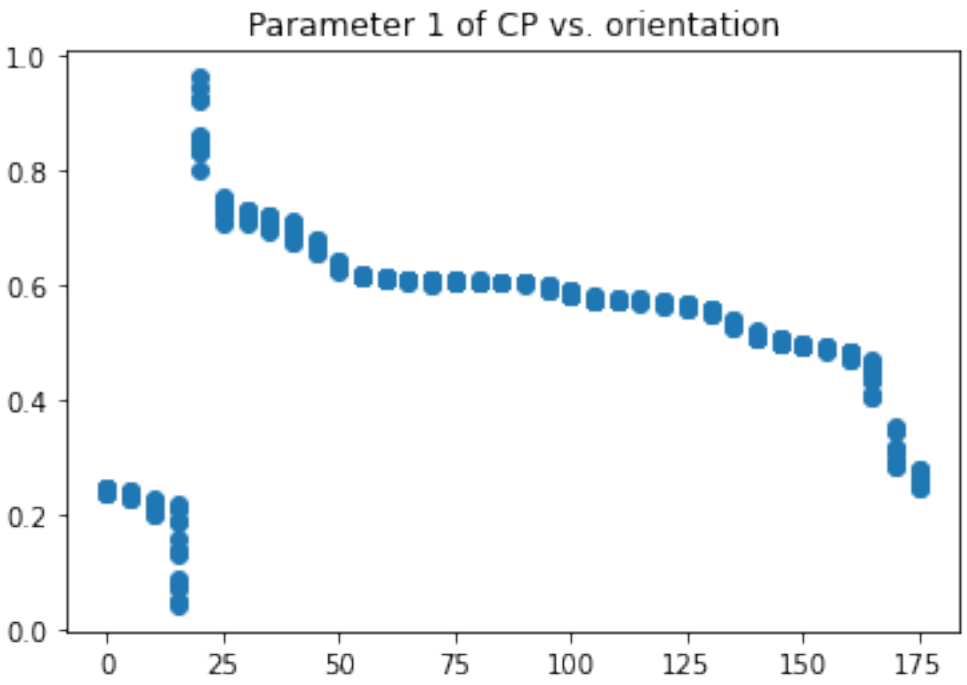
[ ]:
```
<matplotlib.collections.PathCollection at 0x7fefd8cc6fe0>
```



Orientation is decoded:

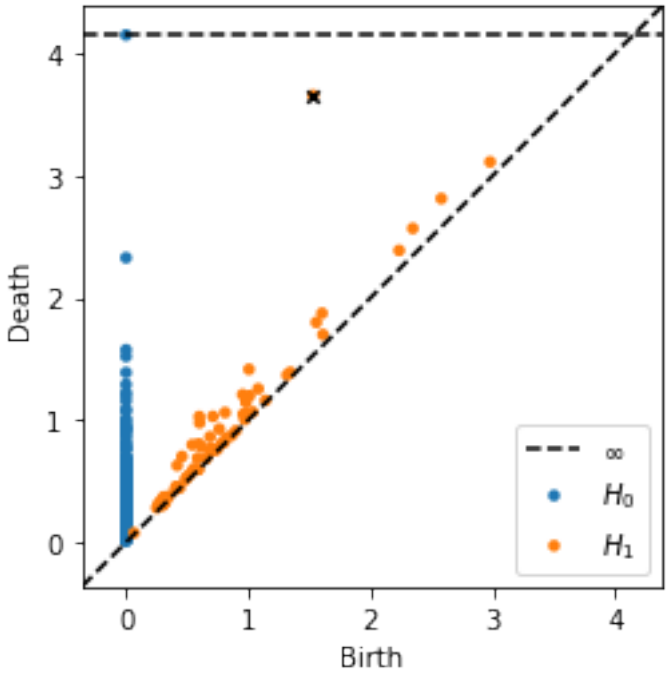
[ ]:
```
plt.title("Parameter 1 of CP vs. orientation")
plt.scatter(theta_deg[::step], param_1)
```

[ ]:
```
<matplotlib.collections.PathCollection at 0x7fefd8cc6230>
```



[ ]:
```
data_without_features = decoding.remove_feature(data_pca[::step].to_numpy(), pd.
↪DataFrame(param_1))
```

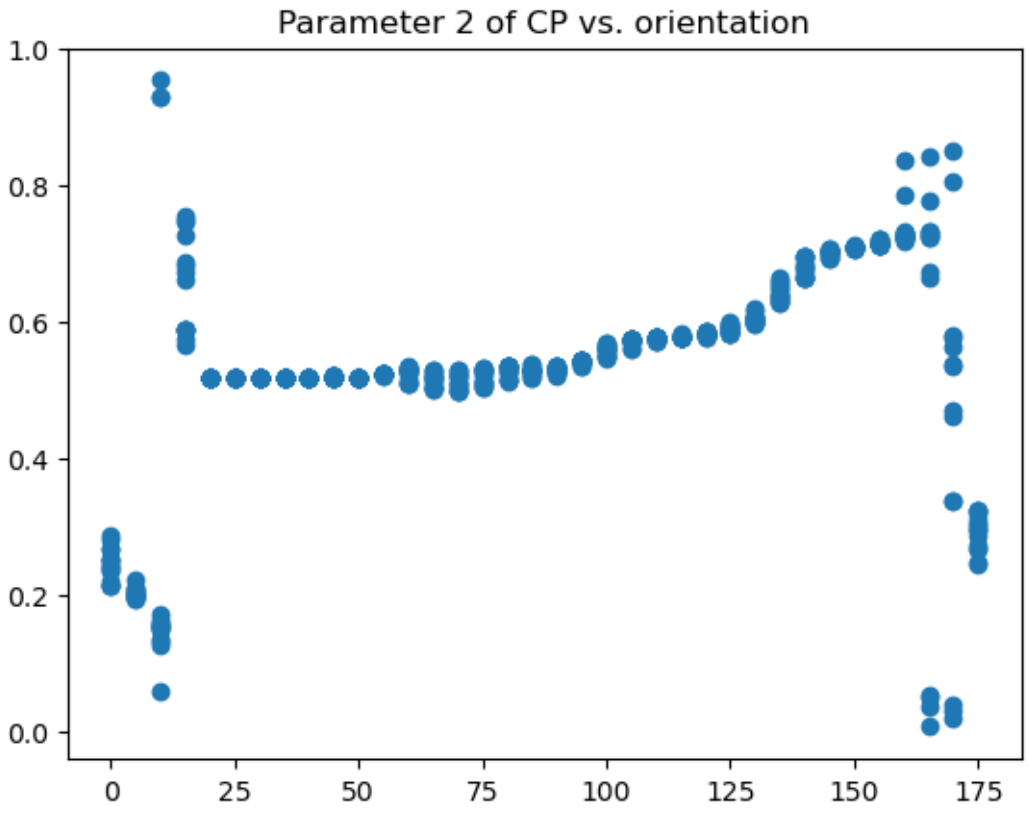
[ ]:
```
param_2 = decoding.cohomological_parameterization(
    pd.DataFrame(data_without_features)).to_numpy()
```




```
Decoding… done
```

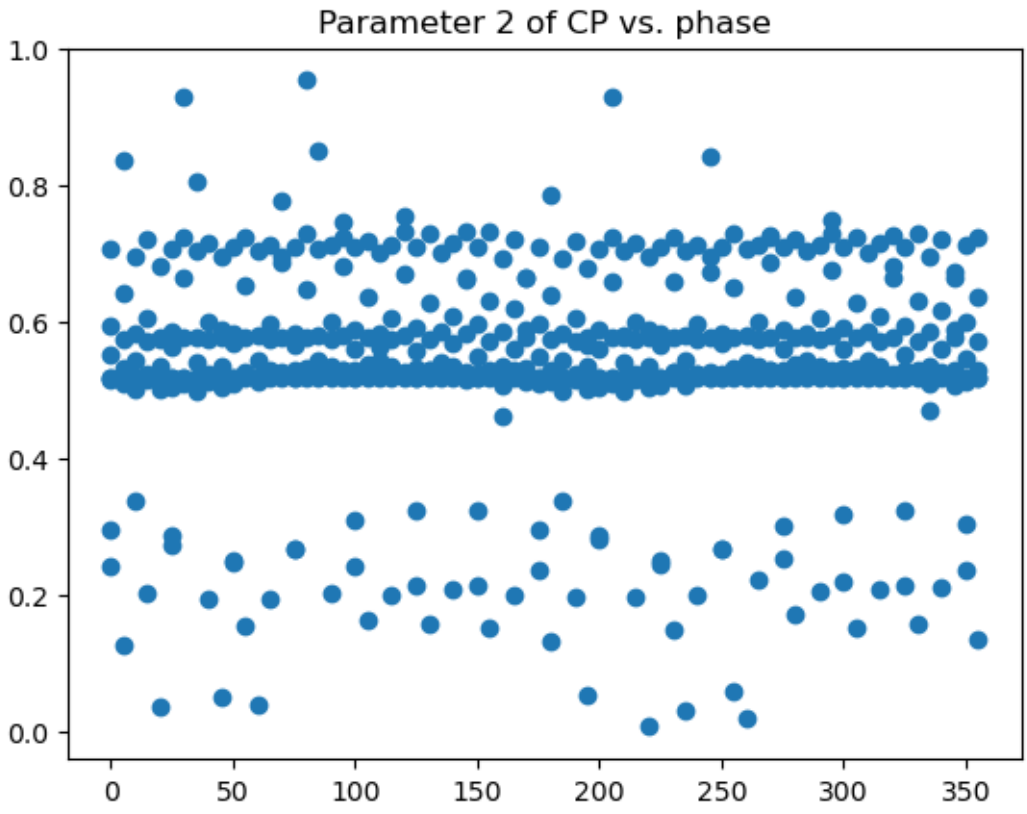
[ ]:
```
plt.title("Parameter 2 of CP vs. orientation")
plt.scatter(theta_deg[::step], param_2)
```

[ ]:
```
<matplotlib.collections.PathCollection at 0x7fefd8be2d40>
```



[ ]:
```
plt.title("Parameter 2 of CP vs. phase")
plt.scatter(phi_deg[::step], param_2)
```

[ ]: <matplotlib.collections.PathCollection at 0x7fefea7e8640>


Parameter 2 of CP vs. phase
1.0
0.8
0.6
0.4
0.2
0.0
0
50
100
150
200
250
300
350