poisson_HMM_research

March 19, 2025

1 $PHMM$ on (monthly, yearly) data agg

1.1 EDA

EDA of the time series to find the earthquakes

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import poisson
from hmmlearn import hmm
```

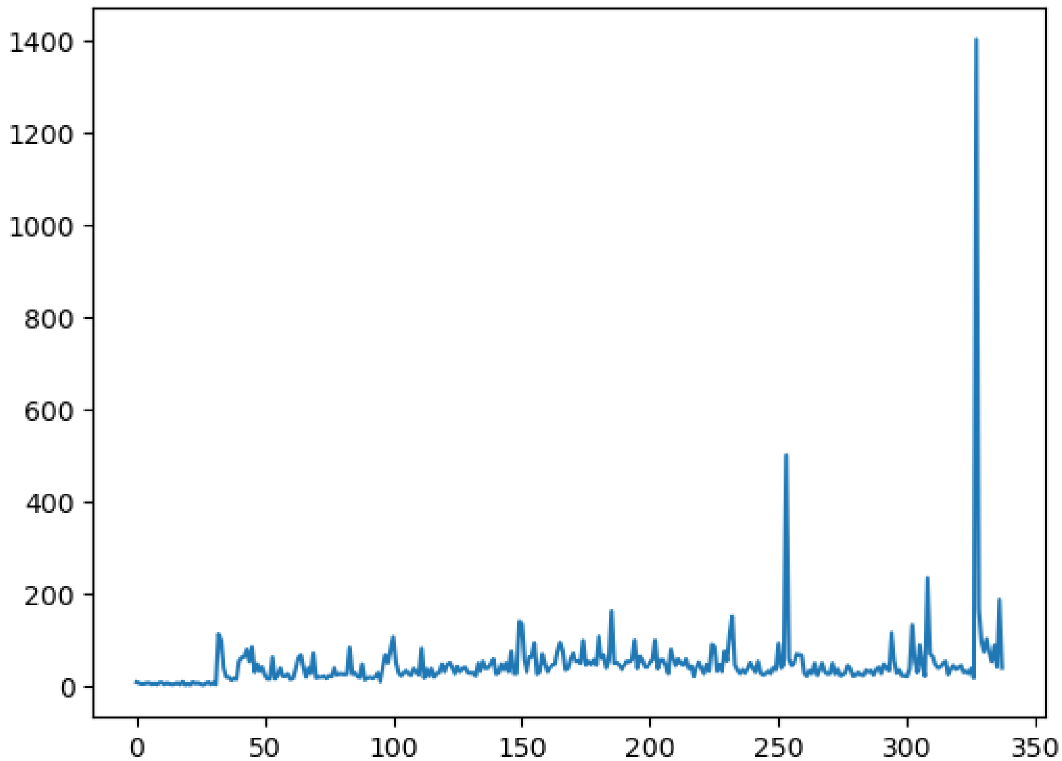
[2]:
```
df = pd.read_csv("All Earthquakes.csv")
df.head(5)
```

[2]:
```
          No.        Orgin date  Longitude(E)  Latitude(N)  Magnitude  Depth  \
0  Small area   2025-02-12 0:09       121.942      24.3840        3.8   24.3
1          64  2025-02-11 22:04       121.665      24.1742        4.0   13.3
2  Small area  2025-02-11 20:46       120.459      23.5483        3.2    5.6
3          63  2025-02-11 15:52       120.496      23.2458        4.1    7.7
4  Small area  2025-02-11 15:45       120.505      23.2862        3.6    7.5

         Location                                        Unnamed: 7  \
0  24.38N 121.94E    i.e. 42.5 km SSE of Yilan County(24.38N 121.94E
1  24.17N 121.66E  i.e. 20.8 km NNE of Hualien County(24.17N 121…
2  23.55N 120.46E  i.e. 19.6 km ENE of Chiayi County(23.55N 120.46E
3  23.25N 120.50E      i.e. 42.5 km NE of Tainan City(23.25N 120.50E
4  23.29N 120.50E      i.e. 46.2 km NE of Tainan City(23.29N 120.50E

                                Unnamed: 8 Unnamed: 9 Unnamed: 10
0     i.e. 42.5 km SSE of Yilan County)        NaN         NaN
1   i.e. 20.8 km NNE of Hualien County)        NaN         NaN
2    i.e. 19.6 km ENE of Chiayi County)        NaN         NaN
3       i.e. 42.5 km NE of Tainan City)        NaN         NaN
4       i.e. 46.2 km NE of Tainan City)        NaN         NaN
```

[3]:
```
years = pd.to_datetime(df['Orgin date']).dt.year
months = pd.to_datetime(df['Orgin date']).dt.month
```

```
dates = pd.DataFrame({'years': years, 'months': months})

counts = dates.groupby(['years', 'months']).size()
plt.plot(np.array(counts))
```

[3]: [<matplotlib.lines.Line2D at 0x29935bc5dc0>]



1.2 Poisson Model fitting

[4]:
```
scores = list()
models = list()

earthquakes = np.array(counts).reshape(-1,1)

for n_components in range(1, 5):
    for idx in range(10):  # ten different random starting states
        # define our hidden Markov model
        model = hmm.PoissonHMM(n_components=n_components, random_state=idx,
                               n_iter=10)
        model.fit(earthquakes)
        models.append(model)
```

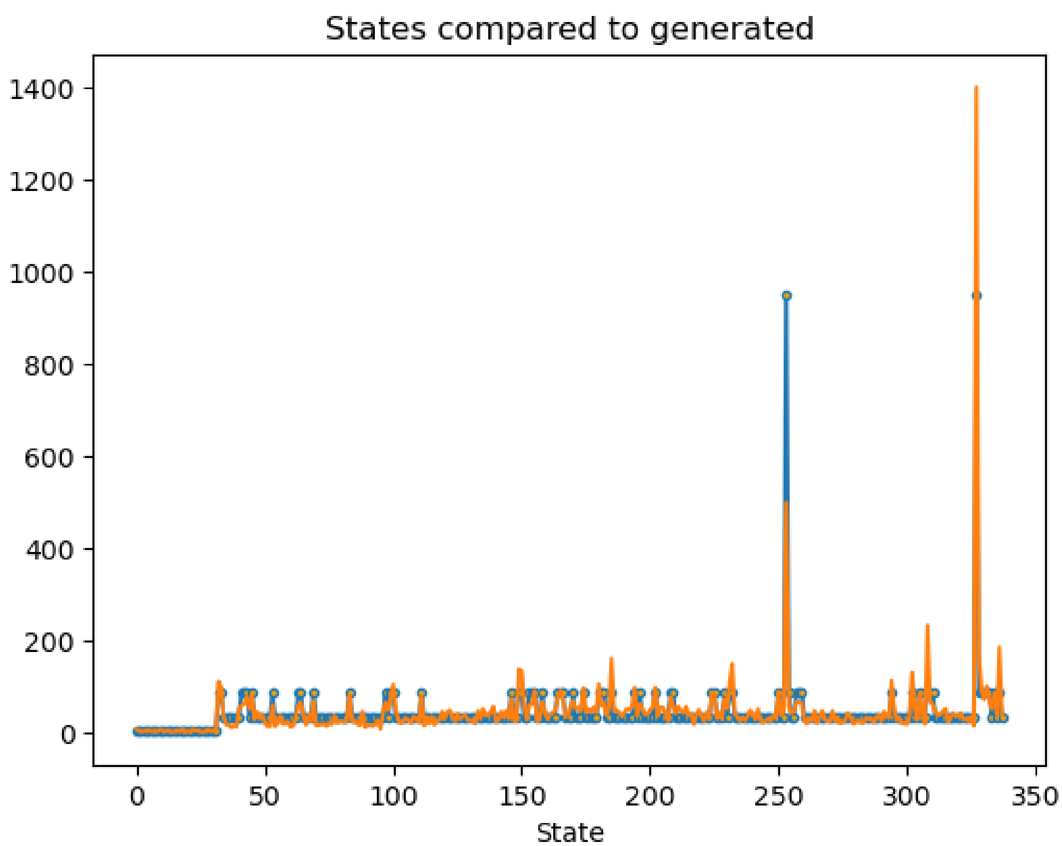
```python
        scores.append(model.score(earthquakes))
        print(f'Converged: {model.monitor_.converged}\t\t'
              f'Score: {scores[-1]}')

# get the best model
model = models[np.argmax(scores)]
print(f'The best model had a score of {max(scores)} and '
      f'{model.n_components} components')

# use the Viterbi algorithm to predict the most likely sequence of states
# given the model
states = model.predict(earthquakes)
```

```
Converged: True         Score: -8241.189334915101
Converged: True         Score: -8241.189334915101
Converged: True         Score: -8241.189334915101
Converged: True         Score: -8241.189334915101
Converged: True         Score: -8241.189334915101
Converged: True         Score: -8241.189334915101
Converged: True         Score: -8241.189334915101
Converged: True         Score: -8241.189334915101
Converged: True         Score: -8241.189334915101
Converged: True         Score: -8241.189334915101
Converged: True         Score: -4864.019134988552
Converged: True         Score: -5386.493068292093
Converged: True         Score: -4953.480260235184
Converged: True         Score: -5000.530264942573
Converged: True         Score: -4246.546995655866
Converged: True         Score: -4934.384000748901
Converged: True         Score: -4896.771505436192
Converged: True         Score: -5083.039872084082
Converged: True         Score: -4246.546995655925
Converged: True         Score: -5150.054824700415
Converged: True         Score: -2928.491397689784
Converged: True         Score: -4727.313490455204
Converged: True         Score: -4042.8728473956835
Converged: True         Score: -2800.7376825016095
Converged: True         Score: -2688.6487176770115
Converged: True         Score: -4023.454968880487
Converged: True         Score: -2815.319006297583
Converged: True         Score: -4164.020640164266
Converged: True         Score: -2706.077502186983
Converged: True         Score: -4285.369762586141
Converged: True         Score: -2449.04171553053
Converged: True         Score: -4150.869122858616
Converged: True         Score: -3989.7062240558425
Converged: True         Score: -2174.529712164159
Converged: True         Score: -2242.982949012548
```

```
Converged: True         Score: -2479.5997409573074
Converged: True         Score: -2837.9186111739023
Converged: True         Score: -3891.788435502881
Converged: True         Score: -2115.6140033801785
Converged: True         Score: -3791.9708050577115
The best model had a score of -2115.6140033801785 and 4 components
```

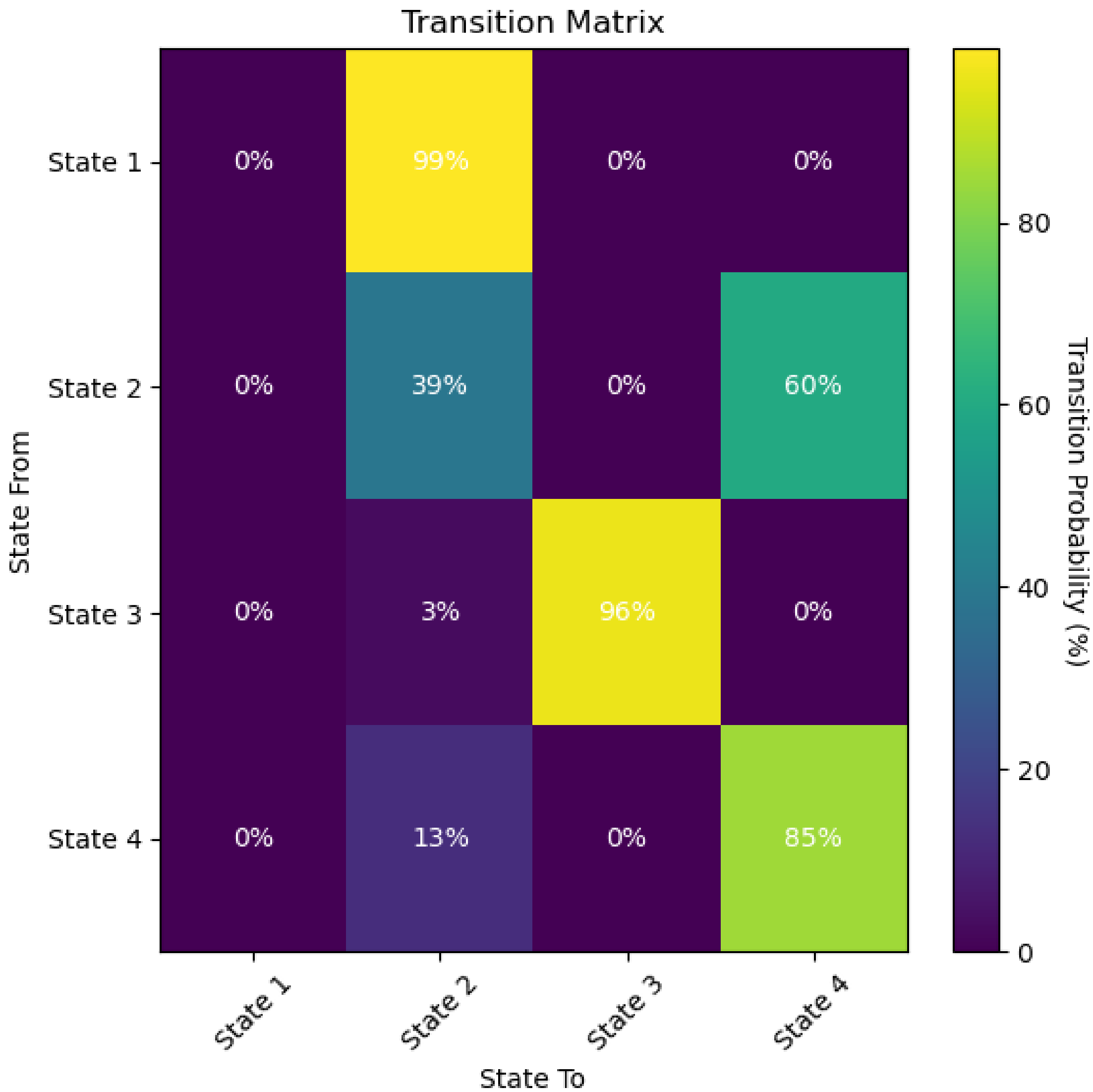
[5]:
```python
fig, ax = plt.subplots()
ax.plot(model.lambdas_[states], ".-", ms=6, mfc="orange")
ax.plot(earthquakes)
ax.set_title('States compared to generated')
ax.set_xlabel('State')
```

[5]:
```
Text(0.5, 0, 'State')
```



1.3 Learned $\widetilde{P}$

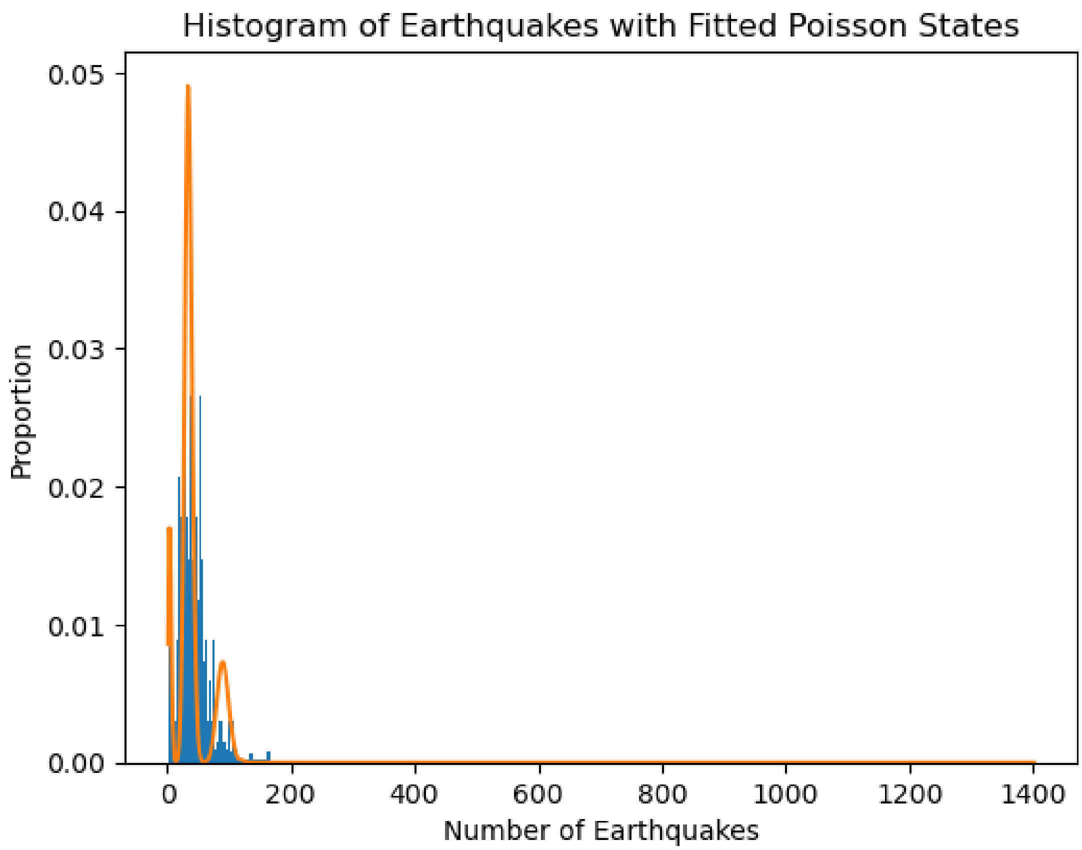
[6]:
```
transmat_int = (model.transmat_ * 100).astype(int)

# Create the heatmap
fig, ax = plt.subplots(figsize=(6, 6))
im = ax.imshow(transmat_int, aspect='auto', cmap='viridis')
cbar = ax.figure.colorbar(im, ax=ax)
cbar.ax.set_ylabel('Transition Probability (%)', rotation=-90, va="bottom")
for i in range(transmat_int.shape[0]):
    for j in range(transmat_int.shape[1]):
        ax.text(j, i, f'{transmat_int[i, j]}%', ha='center', va='center',␣
 ↪color='white')
ax.set_title('Transition Matrix')
ax.set_xlabel('State To')
ax.set_ylabel('State From')
ax.set_xticks(np.arange(transmat_int.shape[1]))
ax.set_yticks(np.arange(transmat_int.shape[0]))
ax.set_xticklabels([f'State {i+1}' for i in range(transmat_int.shape[1])])
ax.set_yticklabels([f'State {i+1}' for i in range(transmat_int.shape[0])])
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



1.4 Plot of fitted poisson states

[7]:
```
prop_per_state = model.predict_proba(earthquakes).mean(axis=0)

# earthquake counts to plot
bins = sorted(np.unique(earthquakes))

fig, ax = plt.subplots()
ax.hist(earthquakes, bins=bins, density=True)
ax.plot(bins, poisson.pmf(bins, model.lambdas_).T @ prop_per_state)
ax.set_title('Histogram of Earthquakes with Fitted Poisson States')
ax.set_xlabel('Number of Earthquakes')
```

```
ax.set_ylabel('Proportion')

plt.show()
```



2 $PHMM$ on yearly data agg

[8]:
```
counts = dates.groupby(['years']).size()
scores = list()
models = list()

earthquakes = np.array(counts).reshape(-1,1)

for n_components in range(1, 5):
    for idx in range(10):  # ten different random starting states
        # define our hidden Markov model
        model = hmm.PoissonHMM(n_components=n_components, random_state=idx,
                               n_iter=10)
        model.fit(earthquakes)
        models.append(model)
```

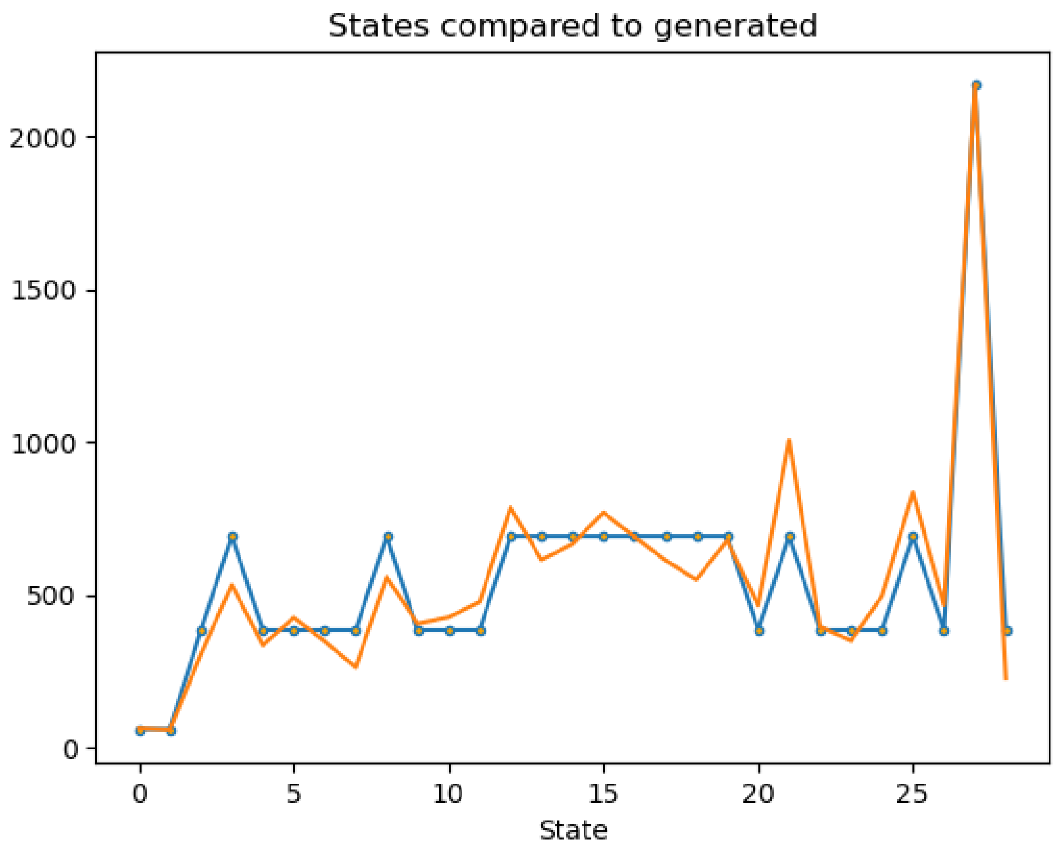
```python
        scores.append(model.score(earthquakes))
        print(f'Converged: {model.monitor_.converged}\t\t'
              f'Score: {scores[-1]}')

# get the best model
model = models[np.argmax(scores)]
print(f'The best model had a score of {max(scores)} and '
      f'{model.n_components} components')

# use the Viterbi algorithm to predict the most likely sequence of states
# given the model
states = model.predict(earthquakes)
```

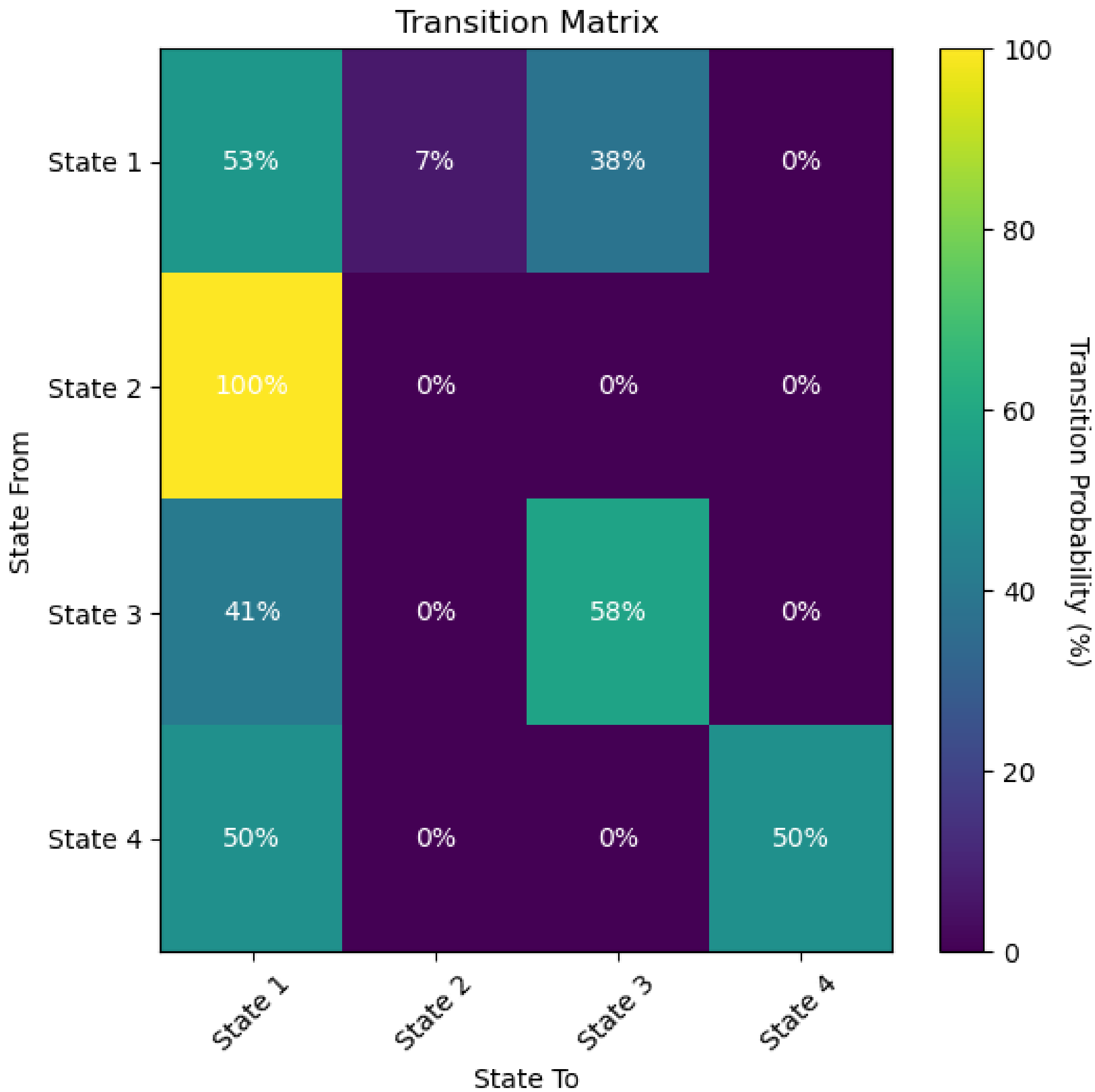
```
Converged: True         Score: -3057.3438840859985
Converged: True         Score: -3057.3438840859985
Converged: True         Score: -3057.3438840859985
Converged: True         Score: -3057.3438840859985
Converged: True         Score: -3057.3438840859985
Converged: True         Score: -3057.3438840859985
Converged: True         Score: -3057.3438840859985
Converged: True         Score: -3057.3438840859985
Converged: True         Score: -3057.3438840859985
Converged: True         Score: -3057.3438840859985
Converged: True         Score: -1618.0196587330643
Converged: True         Score: -1630.1023120958937
Converged: True         Score: -1631.0494194735643
Converged: True         Score: -1618.019658733063
Converged: True         Score: -1631.0494194735638
Converged: True         Score: -1631.0494194735652
Converged: True         Score: -1631.0494194735647
Converged: True         Score: -1631.0494194735652
Converged: True         Score: -1635.6965225437214
Converged: True         Score: -1631.0494194735643
Converged: True         Score: -1046.7857757745674
Converged: True         Score: -905.0369349119644
Converged: True         Score: -1150.1792239807191
Converged: True         Score: -793.0840521842683
Converged: True         Score: -1083.9607897655508
Converged: True         Score: -793.0840521842767
Converged: True         Score: -905.0369349119685
Converged: True         Score: -1150.1792240730929
Converged: True         Score: -1096.974981132629
Converged: True         Score: -1083.9607897655544
Converged: True         Score: -556.4978422156537
Converged: True         Score: -406.76776644643155
Converged: True         Score: -617.2150190910791
Converged: True         Score: -446.89426193256895
Converged: True         Score: -435.61395833603507
```

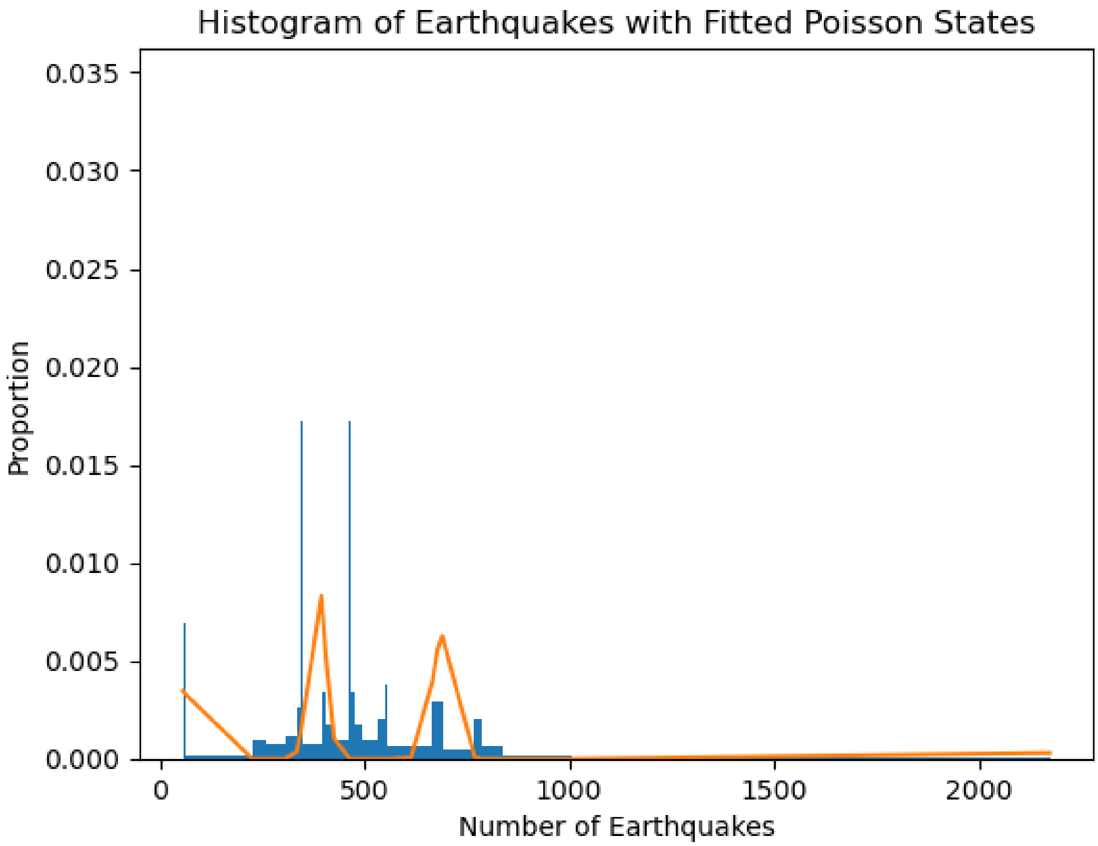
```
Converged: True         Score: -446.8942619249151
Converged: True         Score: -405.30299681994836
Converged: True         Score: -405.3029979038717
Converged: True         Score: -434.65858296279
Converged: True         Score: -418.6794954974542
The best model had a score of -405.30299681994836 and 4 components
```

[9]:
```python
fig, ax = plt.subplots()
ax.plot(model.lambdas_[states], ".-", ms=6, mfc="orange")
ax.plot(earthquakes)
ax.set_title('States compared to generated')
ax.set_xlabel('State')


transmat_int = (model.transmat_ * 100).astype(int)

# Create the heatmap
fig, ax = plt.subplots(figsize=(6, 6))
im = ax.imshow(transmat_int, aspect='auto', cmap='viridis')
cbar = ax.figure.colorbar(im, ax=ax)
cbar.ax.set_ylabel('Transition Probability (%)', rotation=-90, va="bottom")
for i in range(transmat_int.shape[0]):
    for j in range(transmat_int.shape[1]):
        ax.text(j, i, f'{transmat_int[i, j]}%', ha='center', va='center',␣
 ↪color='white')
ax.set_title('Transition Matrix')
ax.set_xlabel('State To')
ax.set_ylabel('State From')
ax.set_xticks(np.arange(transmat_int.shape[1]))
ax.set_yticks(np.arange(transmat_int.shape[0]))
ax.set_xticklabels([f'State {i+1}' for i in range(transmat_int.shape[1])])
ax.set_yticklabels([f'State {i+1}' for i in range(transmat_int.shape[0])])
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()


prop_per_state = model.predict_proba(earthquakes).mean(axis=0)

# earthquake counts to plot
bins = sorted(np.unique(earthquakes))

fig, ax = plt.subplots()
ax.hist(earthquakes, bins=bins, density=True)
ax.plot(bins, poisson.pmf(bins, model.lambdas_).T @ prop_per_state)
ax.set_title('Histogram of Earthquakes with Fitted Poisson States')
ax.set_xlabel('Number of Earthquakes')
```

```
ax.set_ylabel('Proportion')

plt.show()
```




Transition Matrix
State 1
State 2
State 3
State 4
53%
7%
38%
0%
100%
0%
0%
0%
41%
0%
58%
0%
50%
0%
0%
50%
State From
State To
Transition Probability (%)
100
80
60
40
20
0



Histogram of Earthquakes with Fitted Poisson States
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
Proportion
0
500
1000
1500
2000
Number of Earthquakes