Tutorial:

Implementing the Gaussian-Linear Hidden Markov Model (GLHMM), with a package in Python for brain data analysis.

https://glhmm.readthedocs.io/en/latest/index.html

https://github.com/katejarne/CNS_tutorial_2024_GLHMM

glhmm
GAUSSIAN LINEAR HIDDEN MARKOV MODELS

Organization For Computational Neurosciences

Cecilia Jarne
cecilia.jarne@unq.edu.ar
Twitter: @ceciliajarne

https://cfin.au.dk/cfin-labs-research-groups/comand-computational-modelling-analysis-of-neuronal-dynamics/people

glhmm
GAUSSIAN LINEAR HIDDEN MARKOV MODELS

https://arxiv.org/abs/2312.07151

https://cfin.au.dk/cfin-labs-research-groups/comand-computational-modelling-analysis-of-neuronal-dynamics/people

Diego Vidaurre

glhmm
GAUSSIAN LINEAR HIDDEN MARKOV MODELS

https://arxiv.org/abs/2312.07151

Motivation:

- Brain activity is vastly multidimensional and complex, and can be measured with different modalities.

Motivation:

- Brain activity is vastly multidimensional and complex, and can be measured with different modalities.

- Finding structure within this complexity is crucial to establish meaningful associations to behaviour.

Motivation:

The Hidden Markov Model (HMM), with its capacity for finding dynamic network configurations in a time-resolved manner, has emerged as a general family of models that can be applied to a broad array of scientific questions and applications.

What are?

- The HMM represents a (typically multivariate) signal with rich properties using a reduced number of components (i.e. the HMM states), such that each component explains a specific subset of the signal.

What are?

- The HMM represents a (typically multivariate) signal with rich properties using a reduced number of components (i.e. the HMM states), such that each component explains a specific subset of the signal.

- As opposed to principal or independent component analysis, the components can capture advanced aspects of the signal such as specific patterns of correlation or spectral properties.

Motivation II:

- The HMM can still be used as a dimensionality reduction method.

Motivation II:

- The HMM can still be used as a dimensionality reduction method.

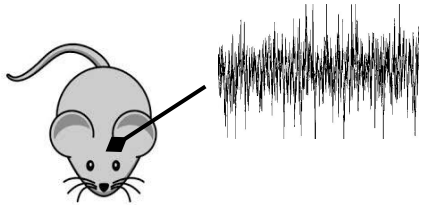
- Brain activity is vastly multidimensional and complex, and can be measured with a plethora of different modalities.

Introduction: HMM as an Unsupervised approach

UNSUPERVISED APPROACHES

Explore what goes on in the brain that does not manifest in behavior

What happens in the brain between stimulus presentation and decision?

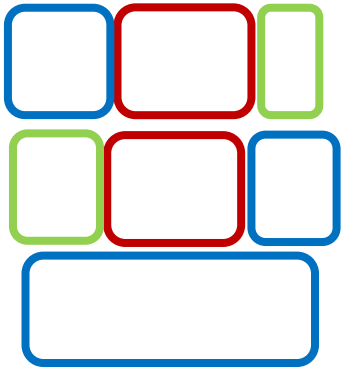


Time-point-by-time-point clustering

Hidden Markov models (HMM)

Find 3 states in the data

Stimulus presentation



Decision

State 1 = anticipation?

State 2 = disengaged ?

State 3 = attentive?

WHY?

Because there is more in the brain than we can observe in the behavior!

Paying attention

Feeling hungry

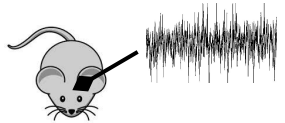
Being bored

Getting surprised

Assumption:
Hidden states the brain goes through

Aim:
Capture these states by examining the dynamics of brain data

SUPERVISED VS UNSUPERVISED METHODS



+ Pros	− Cons	Supervised approaches	
Provide a direct answer on defined data-behavior relation	Require prior info /assumptions on the data-behavior relation	Decoding	Has this mouse performed the task correctly?

https://doi.org/10.1093/cercor/bhab189

Unsupervised approaches	
HMM	What state is the mouse currently in?

Automatically detect states in the data

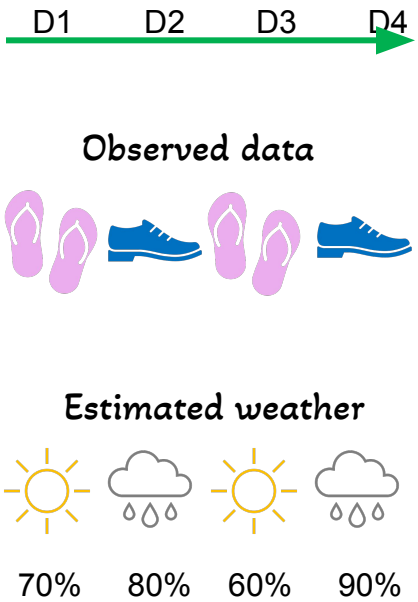
Interpretation of the states is left to the researcher

https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2018.00603/full

WHY HIDDEN MARKOV MODELS (HMM)?

What is a hidden Markov model (HMM)?

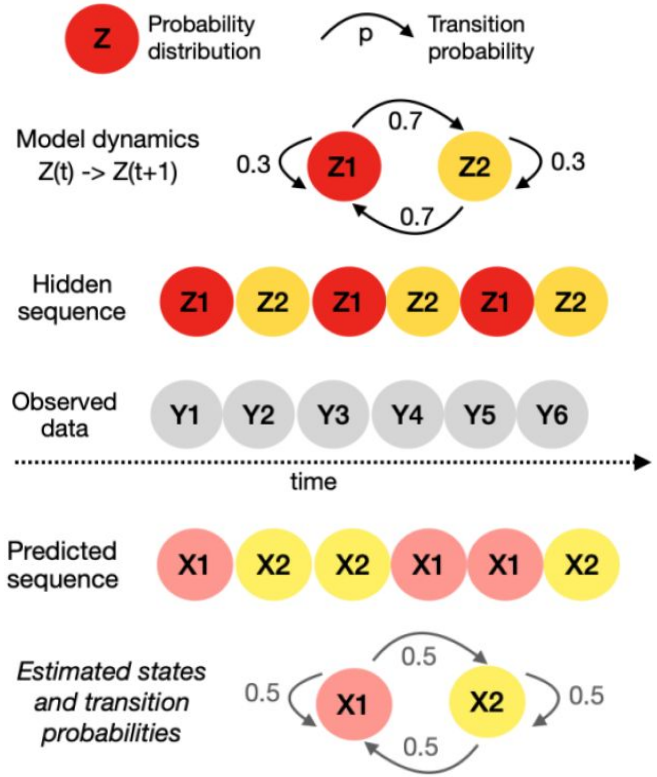
- Unsupervised method
- Deal with time series data
- Estimates the *hidden* distribution of the data (*states*)
- Each time point is assigned to a state
- Probabilistic
- Markov property



WHY HIDDEN MARKOV MODELS (HMM)?

HMM practicalities

- Assume number of states $K \Leftarrow 2$
- Assume probability distribution of states – i.e. X *observation model*
- Get states definition
- Get probability of each data point belonging to a state
- Get probability switching from one state to another

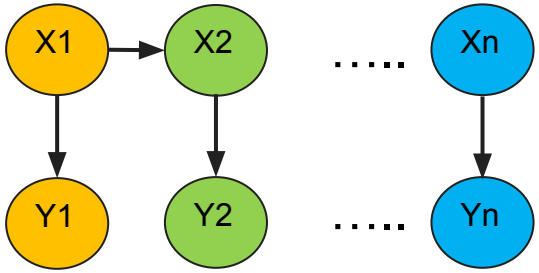


How does HMM work?

Then the HMM uses bayes inference to calculate the probability of the hidden states given the observed data as:

$$P(X \mid Y) = P(Y \mid X)\, P(X) / P(Y) \propto P(X, Y)$$

Prob that data come from states — Prob of states — Prob of data



$$P(X1, X2, \ldots Xn, Y1, Y2, \ldots Yn) = P(X1)\, P(Y1 \mid X1) \prod_{k=2}^{n} P(Yk \mid Xk)\, P(Xk \mid Xk\text{-}1)$$

Prior prob of X1 — Prob that Y1 comes from X1 — Prob of transitioning from state k-1 to state k

The states definition are given by the observation model!

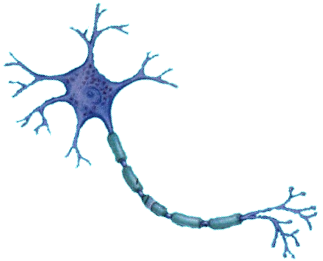
1. First assume the transition probabilities, prior prob of states and emission probability (states generating data) are known.

2. Then use “Bam-Whelch” algorithm (forward-backward algorithm + expectation maximization) to find optimal values for them.

Where can we apply HMMs?

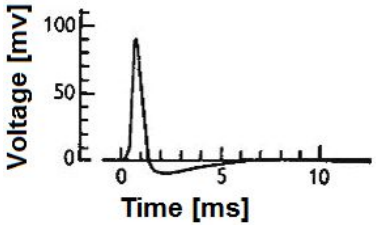
NEURONS (micro scale)

Neuron



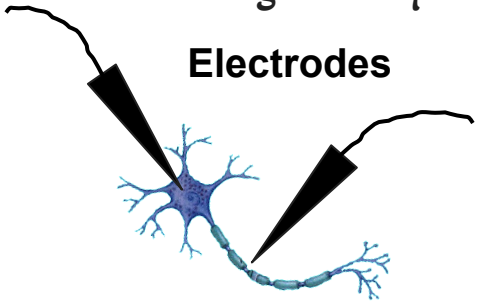
Neurons emit a SPIKE
– or FIRE

Spike



Recording techniques

Electrodes



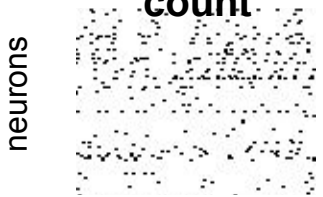
Recording
electrical
activity

Calcium imaging

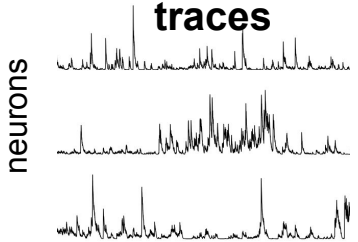
Tracking
chemical
components

Visualization

Spike count

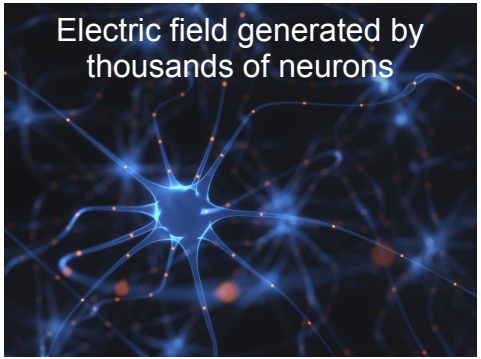


Calcium traces



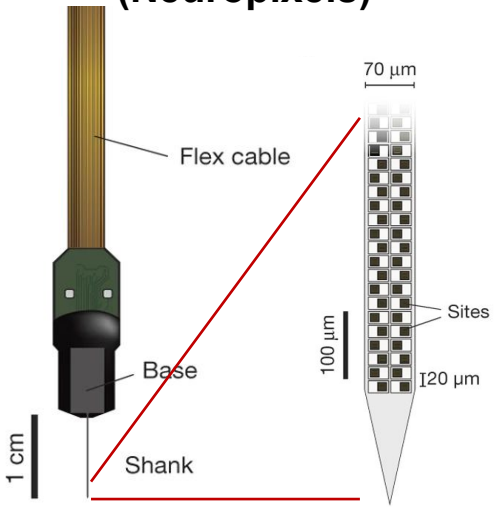
NETWORKS OF NEURONS (meso-scale)

Local Field Potential (LFP)



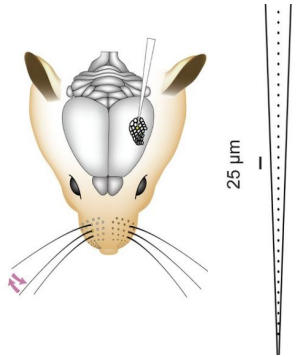
Recording technique

Probes (Neuropixels)

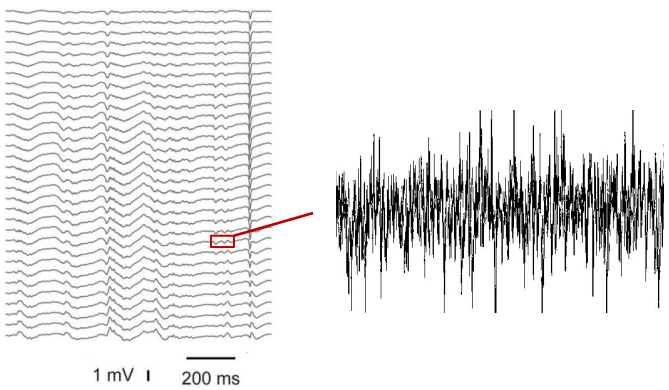


From: Jun *et al.*, 2017

Visualization



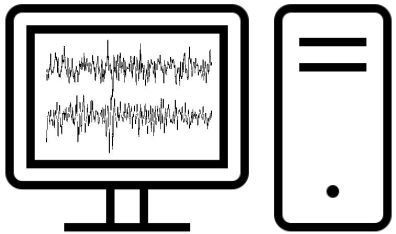
From: Sederberg *et al.*, 2019



WHOLE-BRAIN DATA (macro scale)

Magnetoencephalography (MEG)

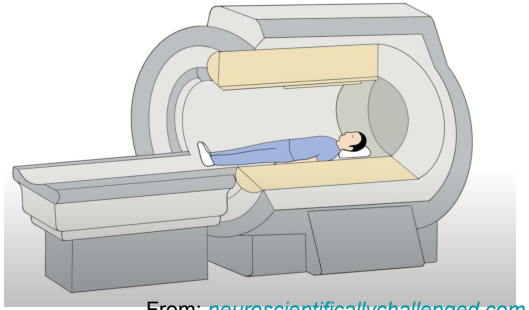
From:
chp.edu



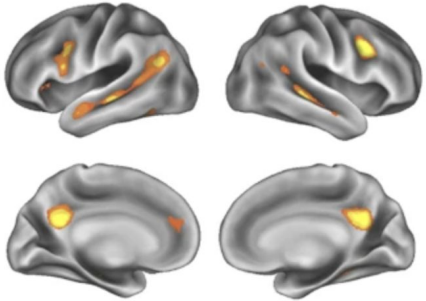
Electroencephalography (EEG)

London, Neurolive Project, Goldsmiths University (2022)

(functional) Magnetic resonance imaging (fMRI)



From: *neuroscientificallychallenged.com*



From: Hoekzema *et al.*, 2017

Standard Gaussian Hidden Markov Model

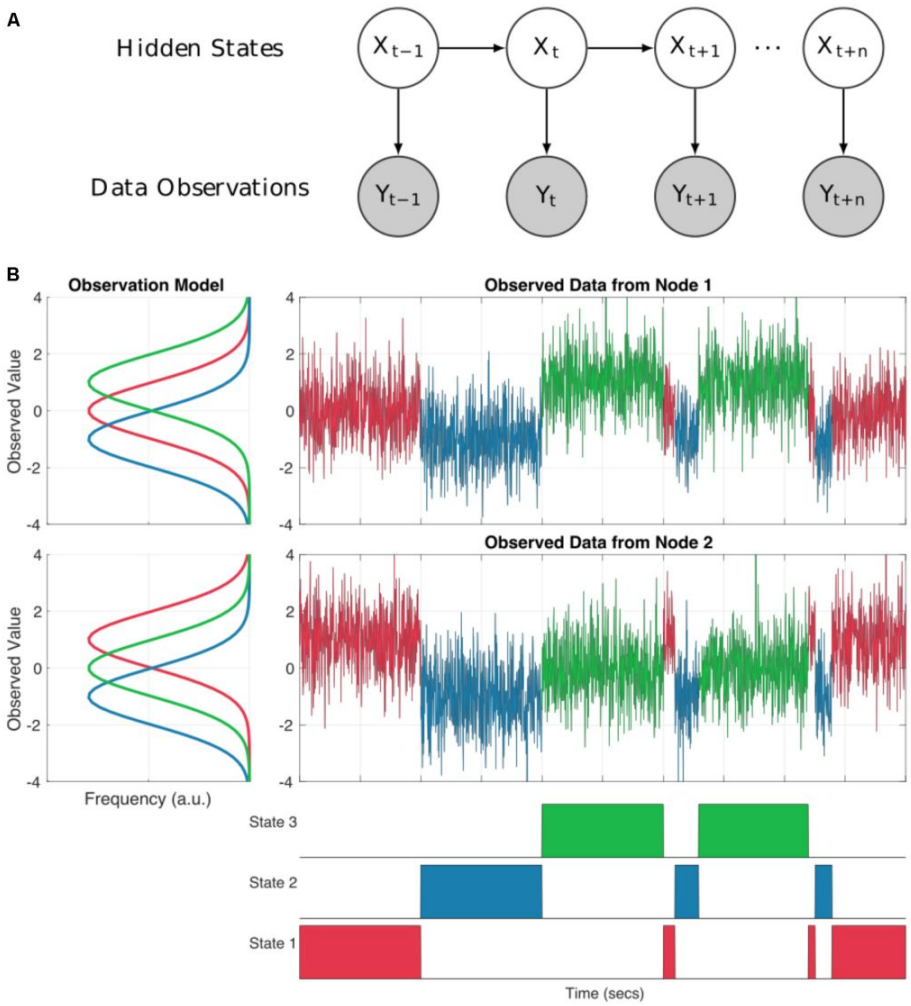
Background:

Standard Gaussian Hidden Markov Model

The HMM is a generative probabilistic model that assumes that an observed time series (e.g., neuroimaging or electrophysiological recordings) was generated by a sequence of hidden "states".

- In the Gaussian HMM, we model states as Gaussian distributions. We assume the observations Y at time point t were generated by a Gaussian distribution with parameters μ and Σ when state k is active, i.e.:

$$Y_t \sim N(\mu^k, \Sigma^k)$$


A
Hidden States
X_{t-1}
X_t
X_{t+1}
X_{t+n}
Data Observations
Y_{t-1}
Y_t
Y_{t+1}
Y_{t+n}
B
Observation Model
Observed Data from Node 1
Observed Data from Node 2
Observed Value
Frequency (a.u.)
State 3
State 2
State 1
Time (secs)

The HMM estimates the probabilities of transitioning θ between each pair of states, i.e., the probability that the currently active state at time point t is k, given that the state at the previous timepoint t-1 was l:

$$P(s_t = k | s_{t-1} = l) = \theta_{k,l}$$

And the probability π that a segment of the timeseries starts with state k:

$$P(s_0 = k) = \pi_k$$

When we fit the model to the observations, we aim to estimate the parameters of the prior distributions for these parameters (μ,Σ ,θ , and π) using variational inference.

We define the posterior estimates as

$$\gamma_{t,k} := P(s_t = k | s_{>t}, s_{<t}, Y)$$

$$\xi_{t,k,l} := P(s_t = k, s_{t-1} = l | s_{>t}, s_{<t-1}, Y)$$

where γ are the probabilities of state k being active at time point t (the state time courses) and are the joint state probabilities. Instead of the state time courses, we can also use the Viterbi path, a discrete representation of which state is active at each time point.

A common application for the standard Gaussian HMM is the estimation of time-varying amplitude and functional connectivity in fMRI recordings (e.g.,Vidaurre et al., 2017).

Viterbi path:

The Viterbi algorithm is a dynamic programming algorithm for obtaining the maximum a posteriori probability estimate of the most likely sequence of hidden states—called the Viterbi path—that results in a sequence of observed events. This is done especially in the context of Markov information sources and hidden Markov models (HMM).

Viterbi path:

The Viterbi algorithm is a dynamic programming algorithm for obtaining the maximum a posteriori probability estimate of the most likely sequence of hidden states—called the Viterbi path—that results in a sequence of observed events. This is done especially in the context of Markov information sources and hidden Markov models (HMM).

The Viterbi path is a discrete representation of the state timecourse, indicating which state is active at which timepoint.

Viterbi path:

The Viterbi algorithm is a dynamic programming algorithm for obtaining the maximum a posteriori probability estimate of the most likely sequence of hidden states—called the Viterbi path—that results in a sequence of observed events. This is done especially in the context of Markov information sources and hidden Markov models (HMM).

The Viterbi path is a discrete representation of the state timecourse, indicating which state is active at which timepoint.

We may be able to see whether some states tend to occur more for certain subjects, or are related to a stimulus that occurs at specific timepoints. The Viterbi path can also be informative to understand whether the HMM is “mixing”, i.e., states occur across subjects, or whether the model estimates the entire session of one subject as one state (see Ahrends et al. 2022).

Example: Modelling time-varying amplitude and functional connectivity in fMRI recordings

Modelling time-varying amplitude and functional connectivity in fMRI recordings

https://glhmm.readthedocs.io/en/latest/notebooks/GaussianHMM_example.html

Modelling time-varying amplitude and functional connectivity in fMRI recordings

https://glhmm.readthedocs.io/en/latest/notebooks/GaussianHMM_example.html

- Example illustrating how to fit and inspect a standard Gaussian HMM.

Modelling time-varying amplitude and functional connectivity in fMRI recordings

https://glhmm.readthedocs.io/en/latest/notebooks/GaussianHMM_example.html

- Example illustrating how to fit and inspect a standard Gaussian HMM.
- The example uses simulated data that can be found in the example_data folder.

Modelling time-varying amplitude and functional connectivity in fMRI recordings

https://glhmm.readthedocs.io/en/latest/notebooks/GaussianHMM_example.html

- Example illustrating how to fit and inspect a standard Gaussian HMM.
- The example uses simulated data that can be found in the example_data folder.
- The data were generated to resemble fMRI timeseries, and our goal is to estimate time-varying amplitude and functional connectivity (FC) for a group of subjects.

Modelling time-varying amplitude and functional connectivity in fMRI recordings

https://glhmm.readthedocs.io/en/latest/notebooks/GaussianHMM_example.html

- Example illustrating how to fit and inspect a standard Gaussian HMM.
- The example uses simulated data that can be found in the example_data folder.
- The data were generated to resemble fMRI timeseries, and our goal is to estimate time-varying amplitude and functional connectivity (FC) for a group of subjects.
- Imagine that the data were recorded from 20 different subjects. Each subject has been recorded for 1,000 timepoints and their timeseries were extracted in a parcellation with 50 brain regions. The data Y here thus has dimensions ((20 subjects * 1000 timepoints), 50 brain regions)

Modelling time-varying amplitude and functional connectivity in fMRI recordings

https://glhmm.readthedocs.io/en/latest/notebooks/GaussianHMM_example.html

- Example illustrating how to fit and inspect a standard Gaussian HMM.
- The example uses simulated data that can be found in the example_data folder.
- The data were generated to resemble fMRI timeseries, and our goal is to estimate time-varying amplitude and functional connectivity (FC) for a group of subjects.
- Imagine that the data were recorded from 20 different subjects. Each subject has been recorded for 1,000 timepoints and their timeseries were extracted in a parcellation with 50 brain regions. The data Y here thus has dimensions ((20 subjects * 1000 timepoints), 50 brain regions)
- We use the indices array to specify where in the timeseries each of the 20 subjects' session starts and ends.

About GLHMM:

About GLHMM:

- GLHMM is a Python toolbox with a focus on neuroscience applications but broadly applicable to other domains as well.

About GLHMM:

- GLHMM is a Python toolbox with a focus on neuroscience applications but broadly applicable to other domains as well.

- It implements a generalisation of various types of Hidden Markov Model (HMM).

About GLHMM:

- GLHMM is a Python toolbox with a focus on neuroscience applications but broadly applicable to other domains as well.

- It implements a generalisation of various types of Hidden Markov Model (HMM).

- Can be applied on multiple data modalities, including fMRI, EEG, MEG, and ECoG, and offers a comprehensive set of HMMs tailored for different data types and analysis goals.

About GLHMM:

- GLHMM is a Python toolbox with a focus on neuroscience applications but broadly applicable to other domains as well.

- It implements a generalisation of various types of Hidden Markov Model (HMM).

- Can be applied on multiple data modalities, including fMRI, EEG, MEG, and ECoG, and offers a comprehensive set of HMMs tailored for different data types and analysis goals.

- The most important configurable aspect is the state distribution, which is parameterized using a regression model.

Steps:

I) Preparation:

If you don't have the GLHMM-package installed, then run the following command in your terminal:

```
pip install glhmm
```

Steps:

I) Preparation:

If you don't have the GLHMM-package installed, then run the following command in your terminal:

```
pip install glhmm
```

Import libraries:
Let’s start by importing the required libraries and modules

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
from glhmm import glhmm, preproc, utils, graphics
```

II) Load data

- The standard HMM requires only one input: a timeseries Y.

- When running the model on a concatenated timeseries, e.g., a group of subjects or several scanning sessions, you also need to provide the indices indicating where each subject/session in the concatenated timeseries Y starts and ends.

- Input data for the glhmm should be in numpy format. Other data types, such as .csv, can be converted to numpy using e.g. pandas, as shown below. Alternatively, the io module provides useful functions to load input data in the required format, e.g., from existing .mat-files. If you need to create indices from session lengths (as used in the HMM-MAR toolbox) you can use the `auxiliary.make_indices_from_T` function.

II) Load data

- The standard HMM requires only one input: a timeseries Y.

- When running the model on a concatenated timeseries, e.g., a group of subjects or several scanning sessions, you also need to provide the indices indicating where each subject/session in the concatenated timeseries Y starts and ends.

- Input data for the glhmm should be in numpy format. Other data types, such as .csv, can be converted to numpy using e.g. pandas, as shown below. Alternatively, the io module provides useful functions to load input data in the required format, e.g., from existing .mat-files. If you need to create indices from session lengths (as used in the HMM-MAR toolbox) you can use the `auxiliary.make_indices_from_T` function.

II) Load data

- The standard HMM requires only one input: a timeseries Y.

- When running the model on a concatenated timeseries, e.g., a group of subjects or several scanning sessions, you also need to provide the indices indicating where each subject/session in the concatenated timeseries Y starts and ends.

- Input data for the glhmm should be in numpy format. Other data types, such as .csv, can be converted to numpy using e.g. pandas, as shown below. Alternatively, the io module provides useful functions to load input data in the required format, e.g., from existing .mat-files. If you need to create indices from session lengths (as used in the HMM-MAR toolbox) you can use the `auxiliary.make_indices_from_T` function.

- Synthetic data for this example are provided in the `glhmm/docs/notebooks/example_data` folder .

- Synthetic data for this example are provided in the `glhmm/docs/notebooks/example_data` folder .

- The file `data.csv` contains synthetic timeseries. The data should have the shape (no subjects/sessions * no timepoints), no features), meaning that all subjects and/or sessions have been concatenated along the first dimension. The second dimension is the number of features, e.g., the number of parcels or channels.

- Synthetic data for this example are provided in the `glhmm/docs/notebooks/example_data` folder .

- The file `data.csv` contains synthetic timeseries. The data should have the shape (no subjects/sessions * no timepoints), no features), meaning that all subjects and/or sessions have been concatenated along the first dimension. The second dimension is the number of features, e.g., the number of parcels or channels.

- The file `T.csv` specifies the indices in the concatenated timeseries corresponding to the beginning and end of individual subjects/sessions in the shape (no subjects, 2). In this case, we have generated timeseries for 20 subjects and 50 features. Each subject has 1,000 timepoints. The timeseries has the shape (20000, 50) and the indices have the shape (20, 2).

```
data = pd.read_csv('./example_data/data.csv', header=None).to_numpy()
 T_t = pd.read_csv('./example_data/T.csv', header=None).to_numpy()
```

- Synthetic data for this example are provided in the `glhmm/docs/notebooks/example_data` folder .

- The file `data.csv` contains synthetic timeseries. The data should have the shape (no subjects/sessions * no timepoints), no features), meaning that all subjects and/or sessions have been concatenated along the first dimension. The second dimension is the number of features, e.g., the number of parcels or channels.

- The file `T.csv` specifies the indices in the concatenated timeseries corresponding to the beginning and end of individual subjects/sessions in the shape (no subjects, 2). In this case, we have generated timeseries for 20 subjects and 50 features. Each subject has 1,000 timepoints. The timeseries has the shape (20000, 50) and the indices have the shape (20, 2).

```
data = pd.read_csv('./example_data/data.csv', header=None).to_numpy()
T_t = pd.read_csv('./example_data/T.csv', header=None).to_numpy()
```

It is important to standardise your timeseries and, if necessary, apply other kinds of preprocessing before fitting the model.

```
data,_ = preproc.preprocess_data(data, T_t)
```

III) Initialise and train an HMM

We first initialise the glhmm object and specify hyperparameters. In the case of the standard Gaussian HMM, since we do not want to model an interaction between two sets of variables, we set model_beta='no'. The number of states is specified using the K parameter. We here estimate K=4 states. If you want to model a different number of states, change K to a different value. In this example, we want to model states as Gaussian distributions with a mean and full covariance matrix, so that each state is described by a mean amplitude and functional connectivity pattern. To do this, specify covtype='full', the state-specific mean is already set by default. If you do not want to model the mean, add model_mean='no'.

```
hmm = glhmm.glhmm(model_beta='no', K=4, covtype='full')
```

III) Initialise and train an HMM

We first initialise the glhmm object and specify hyperparameters. In the case of the standard Gaussian HMM, since we do not want to model an interaction between two sets of variables, we set model_beta='no'. The number of states is specified using the K parameter. We here estimate K=4 states. If you want to model a different number of states, change K to a different value. In this example, we want to model states as Gaussian distributions with a mean and full covariance matrix, so that each state is described by a mean amplitude and functional connectivity pattern. To do this, specify covtype='full', the state-specific mean is already set by default. If you do not want to model the mean, add model_mean='no'.

```
hmm = glhmm.glhmm(model_beta='no', K=4, covtype='full')
```

you can check the hyperparameters (including the ones set by default) to make sure that they correspond to how you want the model to be set up.

```
print(hmm.hyperparameters)
```

We then train the HMM using the data and indices loaded above. Since we here do not model an interaction between two sets of timeseries but run a standard HMM instead, we set X=None. Y should be the timeseries in which we want to estimate states (in here called data) and indices should be the beginning and end indices of each subject (here called T_t). During training, the output will show the progress in model fit at each iteration.

```
hmm.train(X=None, Y=data, indices=T_t)
```

We then train the HMM using the data and indices loaded above. Since we here do not model an interaction between two sets of timeseries but run a standard HMM instead, we set X=None. Y should be the timeseries in which we want to estimate states (in here called data) and indices should be the beginning and end indices of each subject (here called T_t). During training, the output will show the progress in model fit at each iteration.

```
hmm.train(X=None, Y=data, indices=T_t)
```

you can also return Gamma (the state probabilities at each timepoint), Xi (the joint probabilities of past and future states conditioned on the data) and FE (the free energy of each iteration) at this step, but it is also possible to retrieve them later using the decode function for Gamma and Xi and the get_fe function for the free energy.

```
Gamma,Xi,FE = hmm.train(X=None, Y=data, indices=T_t)
```

IV) Inspect model

We can then inspect some interesting aspects of the model.

- We start by checking what the states look like by interrogating the the state means and the state covariances.

- We will then look at the dynamics, i.e., how states transition between each other (transition probabilities) and how the state sequence develops over time (the Viterbi path).

- We will have a look at some summary metrics that can be useful to describe the overall patterns or to relate the model to behaviour using statistical testing or machine learning/out-of-sample prediction.

For more plotting options, see the Graphics module

IV) Inspect model

We can then inspect some interesting aspects of the model.

- We start by checking what the states look like by interrogating the the state means and the state covariances.

- We will then look at the dynamics, i.e., how states transition between each other (transition probabilities) and how the state sequence develops over time (the Viterbi path).

- We will have a look at some summary metrics that can be useful to describe the overall patterns or to relate the model to behaviour using statistical testing or machine learning/out-of-sample prediction.

For more plotting options, see the Graphics module

IV) Inspect model

We can then inspect some interesting aspects of the model.

- We start by checking what the states look like by interrogating the the state means and the state covariances.

- We will then look at the dynamics, i.e., how states transition between each other (transition probabilities) and how the state sequence develops over time (the Viterbi path).

- We will have a look at some summary metrics that can be useful to describe the overall patterns or to relate the model to behaviour using statistical testing or machine learning/out-of-sample prediction.

For more plotting options, see the Graphics module

IV) Inspect model

We can then inspect some interesting aspects of the model.

- We start by checking what the states look like by interrogating the the state means and the state covariances.

- We will then look at the dynamics, i.e., how states transition between each other (transition probabilities) and how the state sequence develops over time (the Viterbi path).

- We will have a look at some summary metrics that can be useful to describe the overall patterns or to relate the model to behaviour using statistical testing or machine learning/out-of-sample prediction.

For more plotting options, see the Graphics module

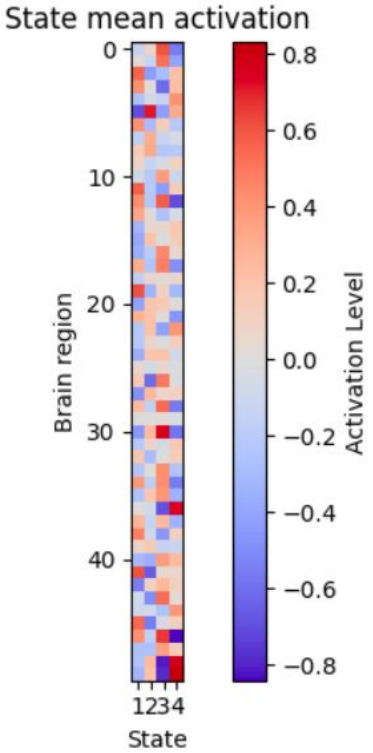
State means: Time-varying amplitude patterns

The state means can be interpreted as time-varying patterns of amplitude (relative to the baseline). They can be retrieved from the model using the get_means() function, or get_mean(k) to retrieve only the mean for state k:

```
K = hmm.hyperparameters["K"] # the number of states
q = data.shape[1] # the number of parcels/channels
state_means = np.zeros(shape=(q, K))
state_means = hmm.get_means() # the state means in the shape (no. features, no. states)
```

We can then plot these amplitude patterns. This will show the states on the x-axis, each parcel/brain region/channel on the y-axis, and the mean activation of each parcel in each state as the color intensity.

```
cmap = "coolwarm"
plt.imshow(state_means, cmap=cmap,interpolation="none")
plt.colorbar(label='Activation Level') # Label for color bar
plt.title("State mean activation")
plt.xticks(np.arange(K), np.arange(1,K+1))
plt.gca().set_xlabel('State')
plt.gca().set_ylabel('Brain region')
plt.tight_layout()  # Adjust layout for better spacing
plt.show()
```



State covariances: Time-varying functional connectivity

The state covariances represent the time-varying functional connectivity patterns that we have estimated in the fMRI recordings. They can be obtained from the model using the `get_covariance_matrix` function:

```
state_FC = np.zeros(shape=(q, q, K))
for k in range(K):
    state_FC[:,:,k] = hmm.get_covariance_matrix(k=k)
```

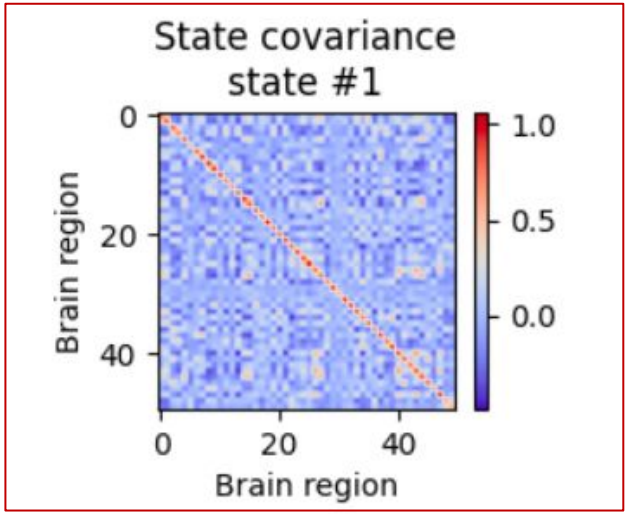
State covariances: Time-varying functional connectivity

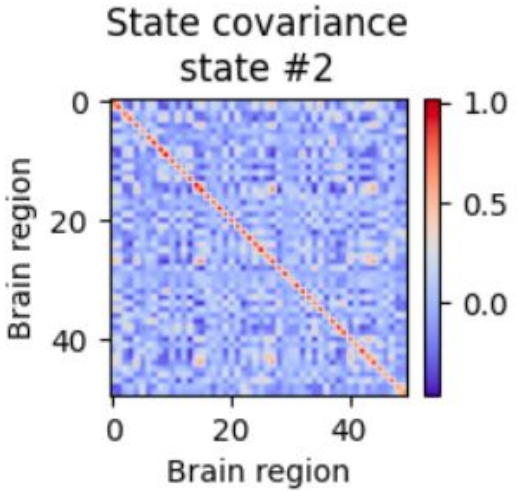
The state covariances represent the time-varying functional connectivity patterns that we have estimated in the fMRI recordings. They can be obtained from the model using the `get_covariance_matrix` function:

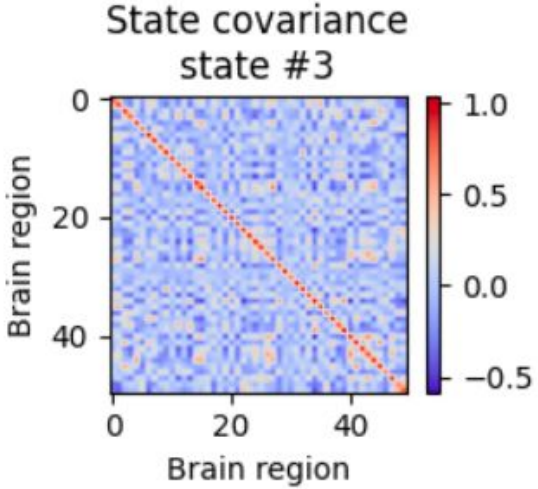
```
state_FC = np.zeros(shape=(q, q, K))
for k in range(K):
    state_FC[:,:,k] = hmm.get_covariance_matrix(k=k)
```

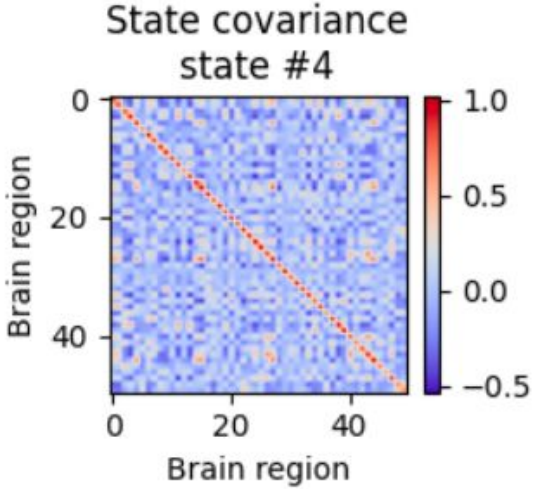
We can then plot the covariance (i.e., functional connectivity) of each state. These are square matrices showing the brain region by brain region functional connectivity patterns:

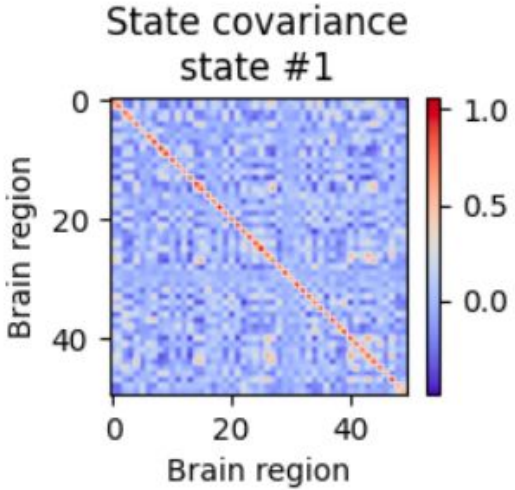
```
for k in range(K):
    plt.subplot(2,2,k+1)
    plt.imshow(state_FC[:,:,k],cmap=cmap)
    plt.xlabel('Brain region')
    plt.ylabel('Brain region')
    plt.colorbar()
    plt.title("State covariance\nstate #%s" % (k+1))
plt.subplots_adjust(hspace=0.7, wspace=0.8)
plt.show()
```

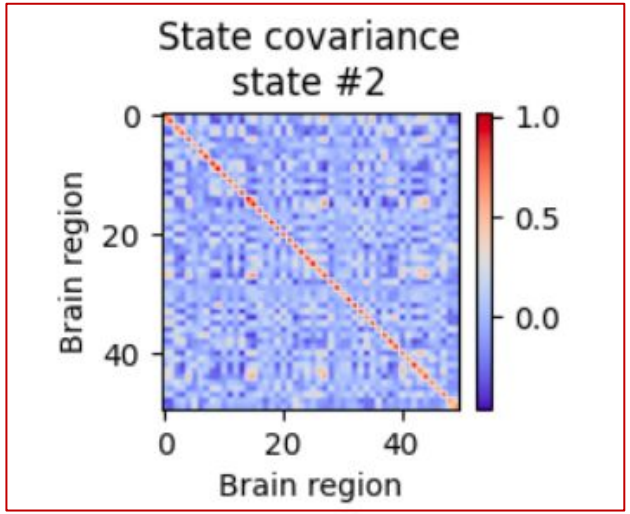

State covariance
state #1
Brain region
Brain region
0
20
40
1.0
0.5
0.0

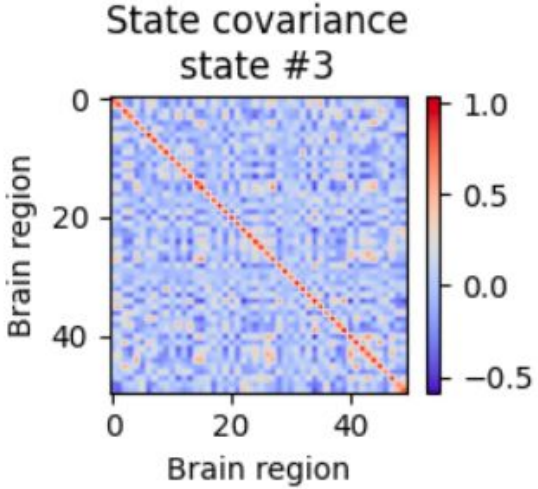

State covariance
state #2
Brain region
Brain region
0
20
40
1.0
0.5
0.0

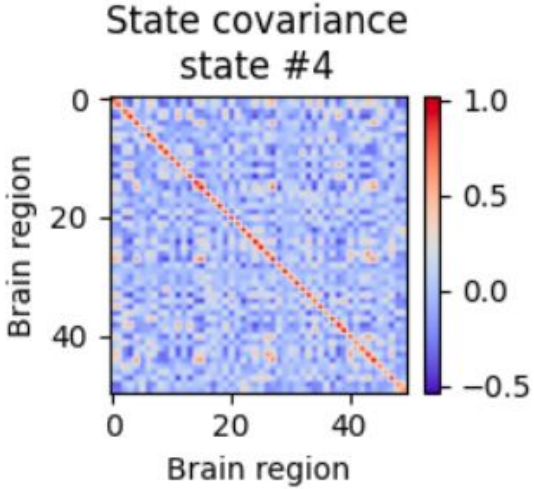

State covariance
state #3
Brain region
Brain region
0
20
40
1.0
0.5
0.0
−0.5

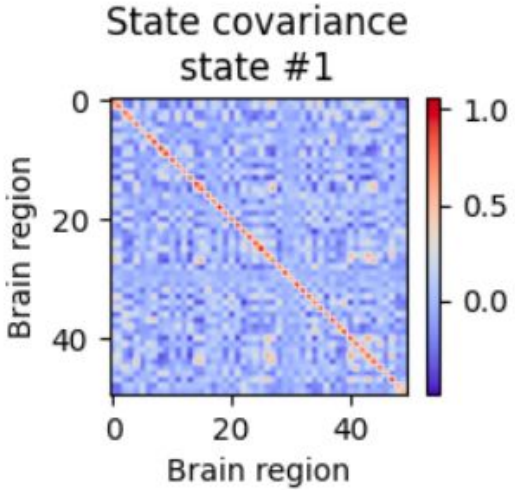

State covariance
state #4
Brain region
Brain region
0
20
40
1.0
0.5
0.0
−0.5

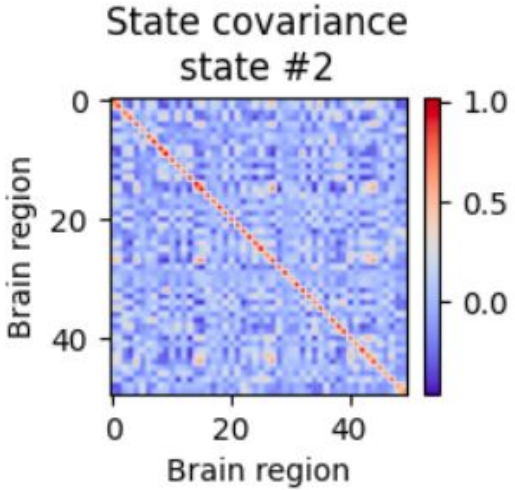

State covariance
state #1
Brain region
Brain region
0
20
40
1.0
0.5
0.0

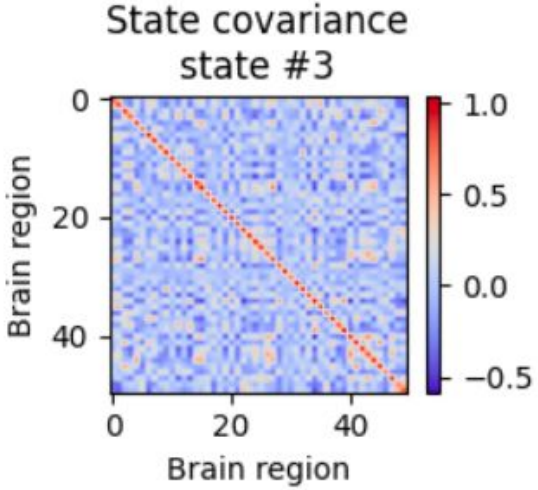

State covariance
state #2
Brain region
Brain region
0
20
40
1.0
0.5
0.0

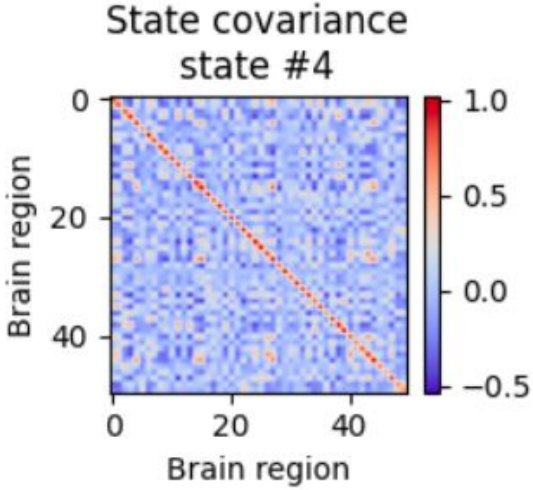

State covariance
state #3
Brain region
Brain region
0
20
40
1.0
0.5
0.0
−0.5

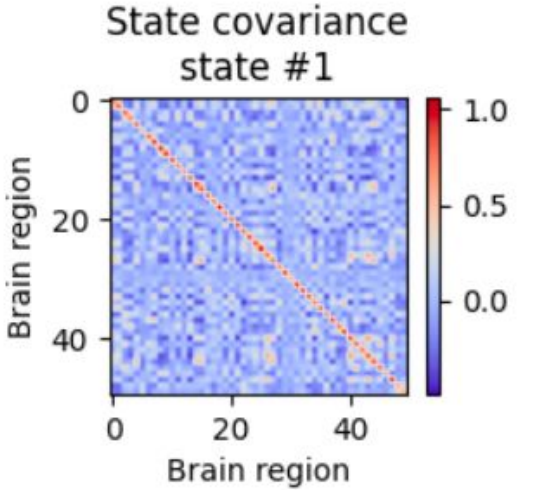

State covariance
state #4
Brain region
Brain region
0
20
40
1.0
0.5
0.0
−0.5

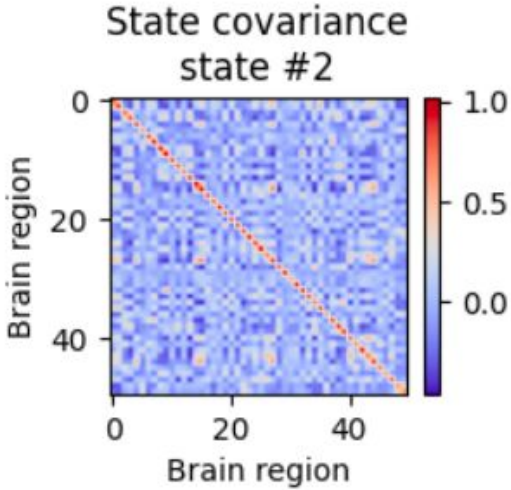

State covariance
state #1
Brain region
Brain region
0
20
40
1.0
0.5
0.0

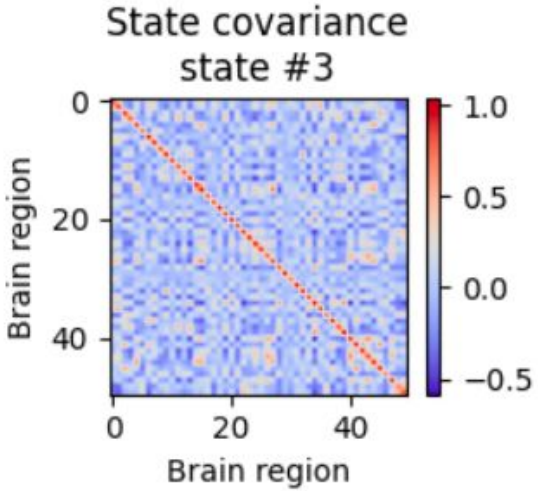

State covariance
state #2
Brain region
Brain region
0
20
40
1.0
0.5
0.0

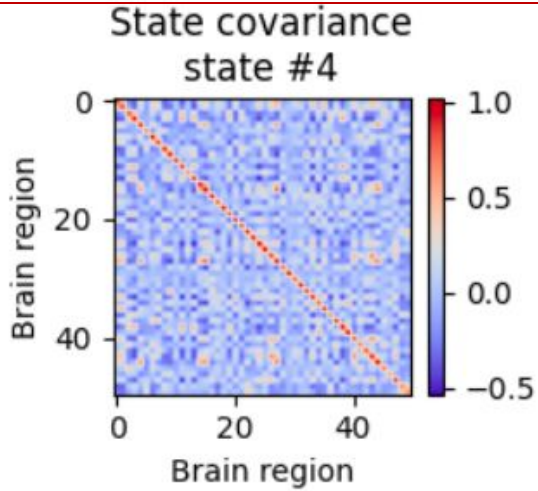

State covariance
state #3
Brain region
Brain region
0
20
40
1.0
0.5
0.0
−0.5


State covariance
state #4
Brain region
Brain region
0
20
40
1.0
0.5
0.0
−0.5


State covariance
state #1
Brain region
Brain region
0
20
40
1.0
0.5
0.0


State covariance
state #2
Brain region
Brain region
0
20
40
1.0
0.5
0.0


State covariance
state #3
Brain region
Brain region
0
20
40
1.0
0.5
0.0
−0.5


State covariance
state #4
Brain region
Brain region
0
20
40
1.0
0.5
0.0
−0.5

Transition probabilities:

The transition probabilities indicate the temporal order in the state sequence, i.e., the probability of transitioning from any one state to any other state. They are contained in `hmm.P` with a shape of [K, K]:

```
TP = hmm.P.copy() # the transition probability matrix
```

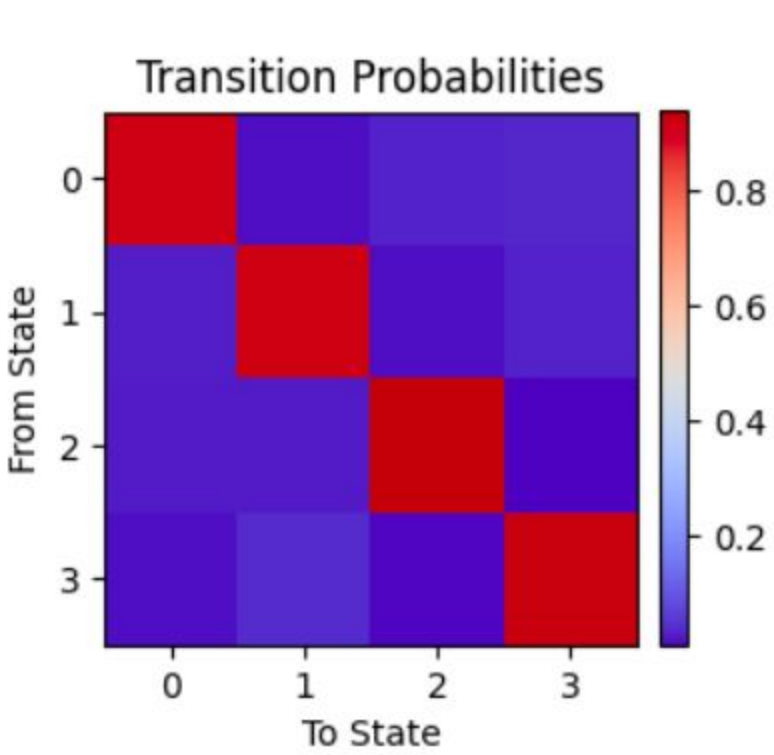
We can then plot the transition probability matrix. Note that self-transitions (i.e., staying in the same state) are considerably more likely in a timeseries that has some order, so there should be a strong diagonal pattern. For comparison, we also show the transition probabilities excluding self-transitions:

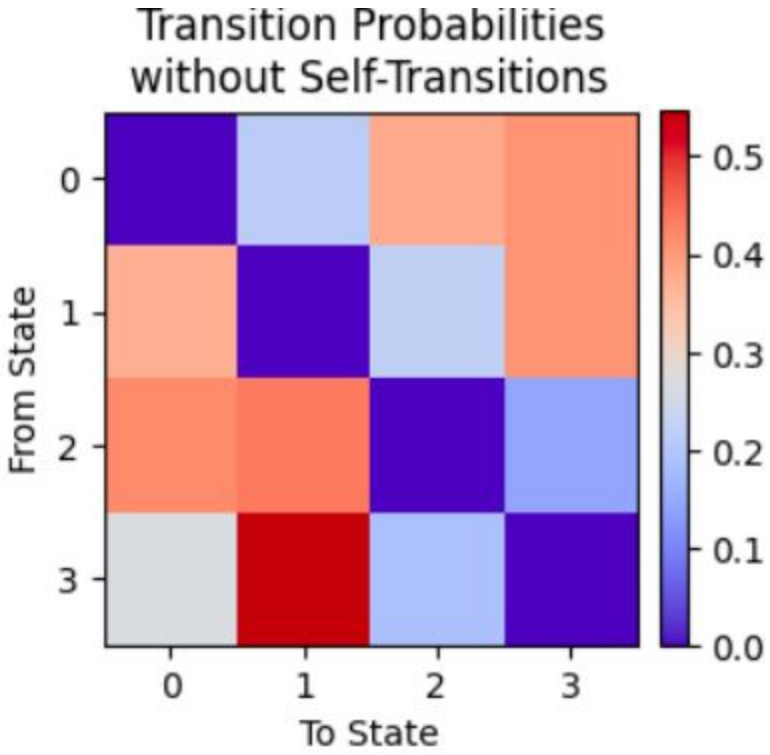
```
# Plot Transition Probabilities
plt.figure(figsize=(7, 4))

# Plot 1: Original Transition Probabilities
plt.subplot(1, 2, 1)
plt.imshow(TP, cmap=cmap, interpolation='nearest')  # Improved color mapping
plt.title('Transition Probabilities')
plt.xlabel('To State')
plt.ylabel('From State')
plt.colorbar(fraction=0.046, pad=0.04)

# Plot 2: Transition Probabilities without Self-Transitions
TP_noself = TP - np.diag(np.diag(TP))  # Remove self-transitions
TP_noself2 = TP_noself / TP_noself.sum(axis=1, keepdims=True)  # Normalize probabilities
plt.subplot(1, 2, 2)
plt.imshow(TP_noself2, cmap=cmap, interpolation='nearest')  # Improved color mapping
plt.title('Transition Probabilities\nwithout Self-Transitions')
plt.xlabel('To State')
plt.ylabel('From State')
plt.colorbar(fraction=0.046, pad=0.04)

plt.tight_layout()  # Adjust layout for better spacing
plt.show()
```


Transition Probabilities
From State
To State
0
1
2
3
0.8
0.6
0.4
0.2

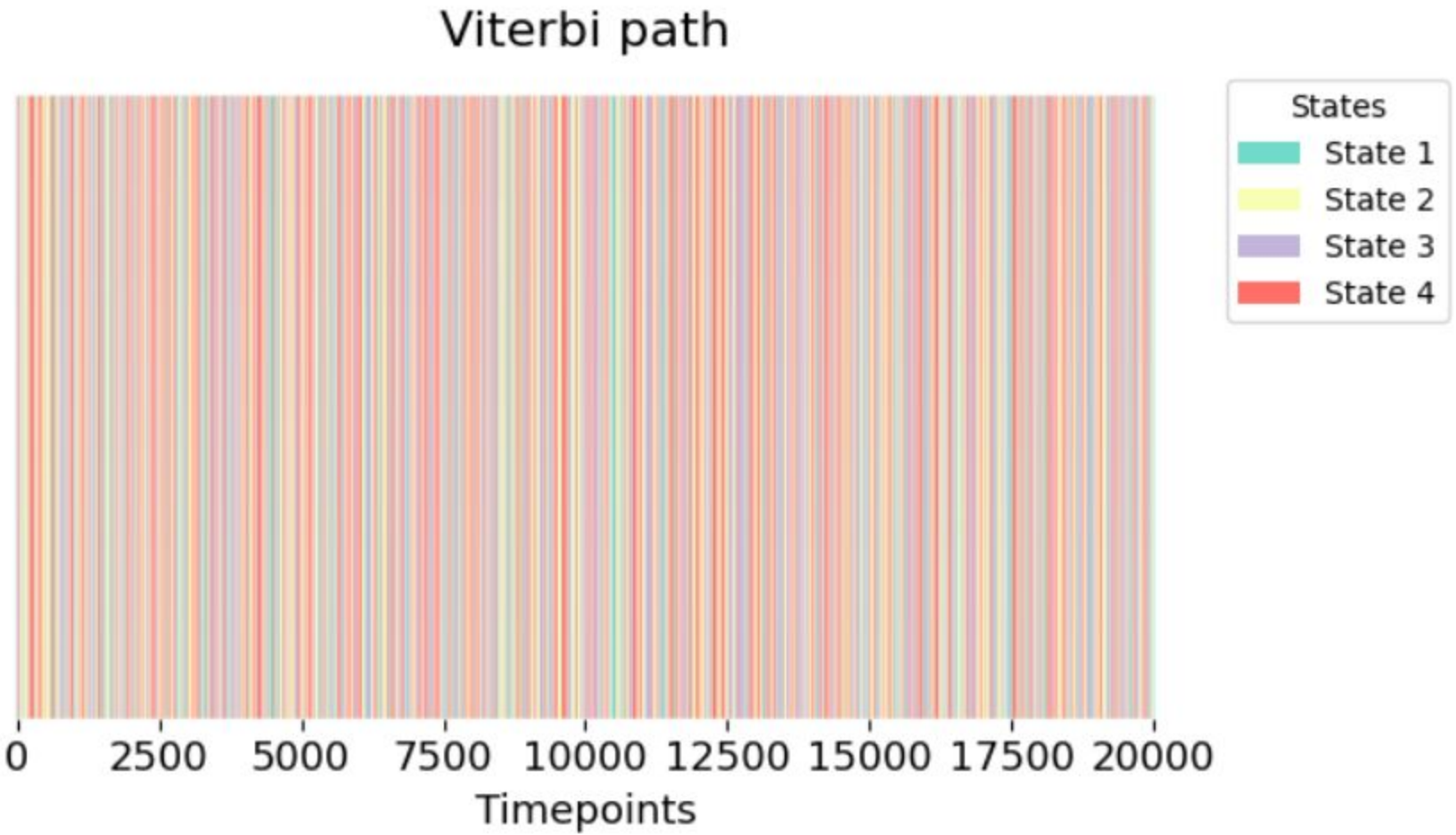

Transition Probabilities
without Self-Transitions
From State
To State
0
1
2
3
0.5
0.4
0.3
0.2
0.1
0.0

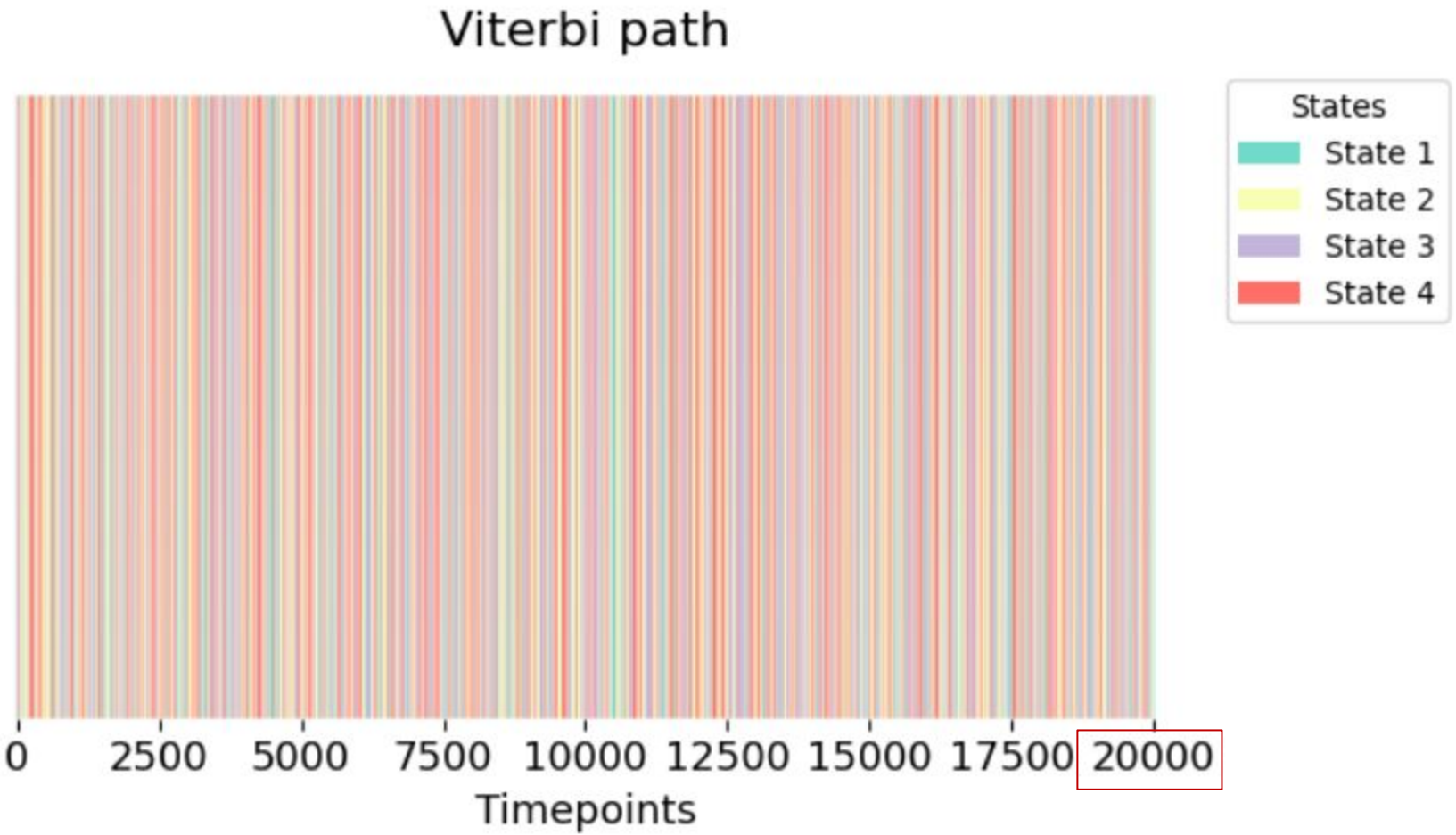
You can retrieve the Viterbi path using the decode function and setting viterbi=True:

```
vpath = hmm.decode(X=None, Y=data, indices=T_t, viterbi=True)
```

And plot the Viterbi path (see also graphics module):

```
graphics.plot_vpath(vpath, title="Viterbi path")
```


Viterbi path
States
State 1
State 2
State 3
State 4
0
2500
5000
7500
10000
12500
15000
17500
20000
Timepoints

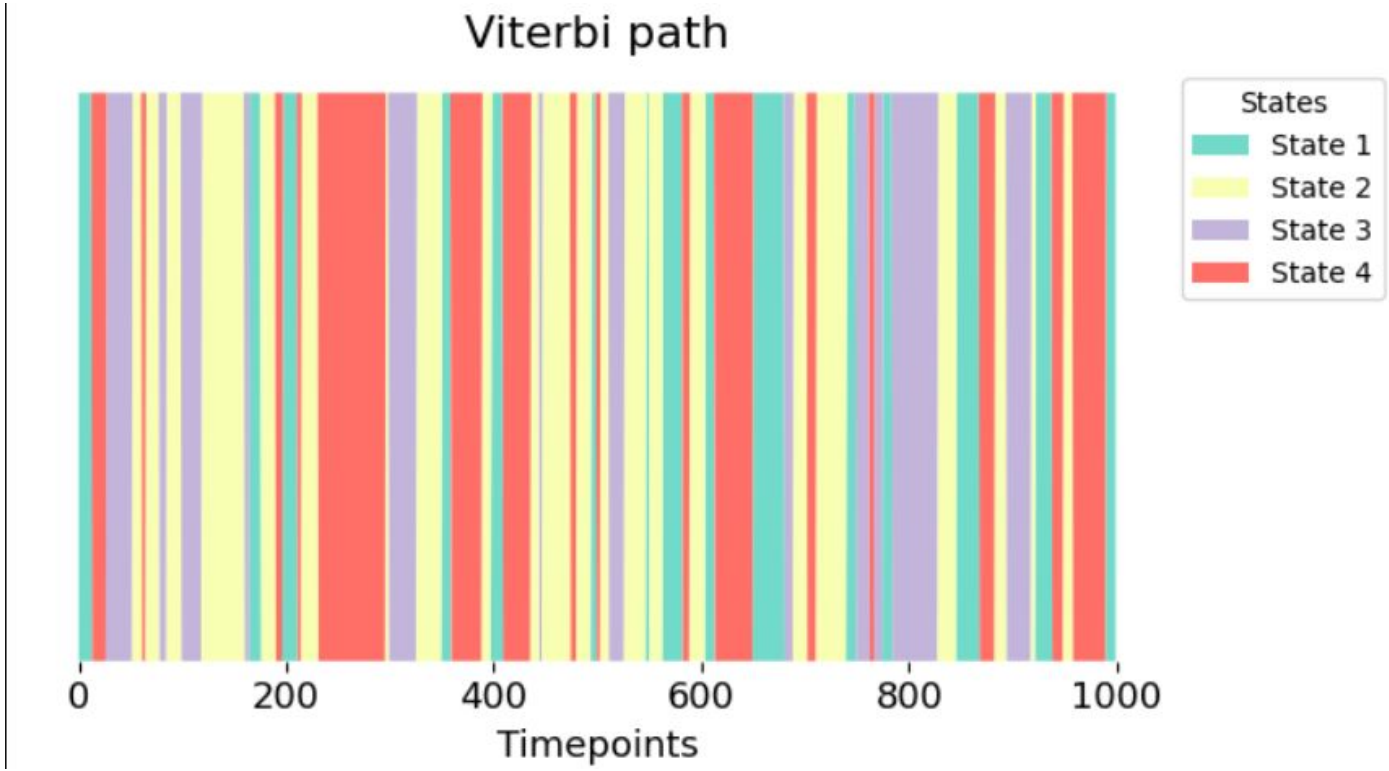

Viterbi path
States
State 1
State 2
State 3
State 4
0
2500
5000
7500
10000
12500
15000
17500
20000
Timepoints

We can also visualize the Viterbi path for the first subject from timepoints 0-1000

```
num_subject = 0
graphics.plot_vpath(vpath[T_t[num_subject,0]:T_t[num_subject,1],:], title="Viterbi path")
```

We can also visualize the Viterbi path for the first subject from timepoints 0-1000

```
num_subject = 0
graphics.plot_vpath(vpath[T_t[num_subject,0]:T_t[num_subject,1],:], title="Viterbi path")
```



Summary metrics:

Once we have estimated the model parameters, we can compute some summary metrics to describe the patterns we see more broadly. These summary metrics can be useful to relate patterns in the timeseries to behaviour, using e.g., statistical testing (see Statistical testing tutorial) or machine learning (see Prediction tutorial). The module utils provides useful functions to obtain these summary metrics.

Summary metrics:

Once we have estimated the model parameters, we can compute some summary metrics to describe the patterns we see more broadly. These summary metrics can be useful to relate patterns in the timeseries to behaviour, using e.g., statistical testing (see Statistical testing tutorial) or machine learning (see Prediction tutorial). The module utils provides useful functions to obtain these summary metrics.

The fractional occupancy (FO) indicates the fraction of time in each session that is occupied by each state. For instance, if one state was active for the entire duration of the session, this state's FO would be 1 (100%) and all others would be 0. If, on the other hand, all states are present for an equal amount of timepoints in total, the FO of all states would be 1/K (the number of states). This can be informative to understand whether one state is more present in a certain group of subjects or experimental condition, or to interrogate mixing (explained above).

Summary metrics:

Once we have estimated the model parameters, we can compute some summary metrics to describe the patterns we see more broadly. These summary metrics can be useful to relate patterns in the timeseries to behaviour, using e.g., statistical testing (see Statistical testing tutorial) or machine learning (see Prediction tutorial). The module utils provides useful functions to obtain these summary metrics.

The fractional occupancy (FO) indicates the fraction of time in each session that is occupied by each state. For instance, if one state was active for the entire duration of the session, this state's FO would be 1 (100%) and all others would be 0. If, on the other hand, all states are present for an equal amount of timepoints in total, the FO of all states would be 1/K (the number of states). This can be informative to understand whether one state is more present in a certain group of subjects or experimental condition, or to interrogate mixing (explained above).

You can obtain the fractional occupancies using the get_FO function. The output is an array containing the FO of each subject along the first dimension and each state along the second dimension.

The switching rate indicates how quickly subjects switch between states (as opposed to stay in the same state).

```
SR = utils.get_switching_rate(Gamma, T_t)
```

And plot the switching rate (see also graphics module):

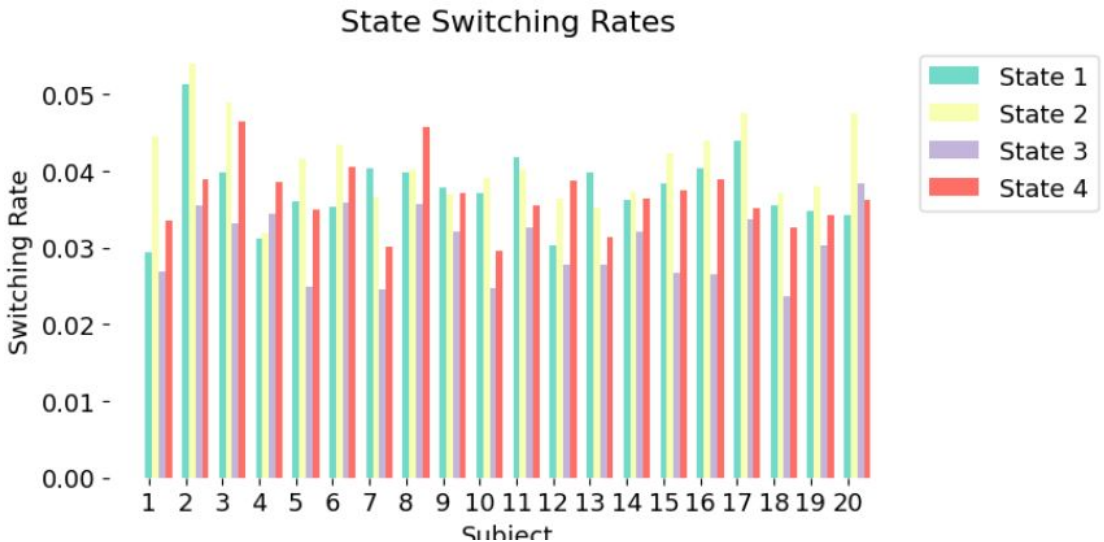
```
graphics.plot_switching_rates(SR, num_ticks=SR.shape[0])
```

The switching rate indicates how quickly subjects switch between states (as opposed to stay in the same state).

```
SR = utils.get_switching_rate(Gamma, T_t)
```

And plot the switching rate (see also graphics module):

```
graphics.plot_switching_rates(SR, num_ticks=SR.shape[0])
```



```
FO = utils.get_FO(Gamma, indices=T_t)
```

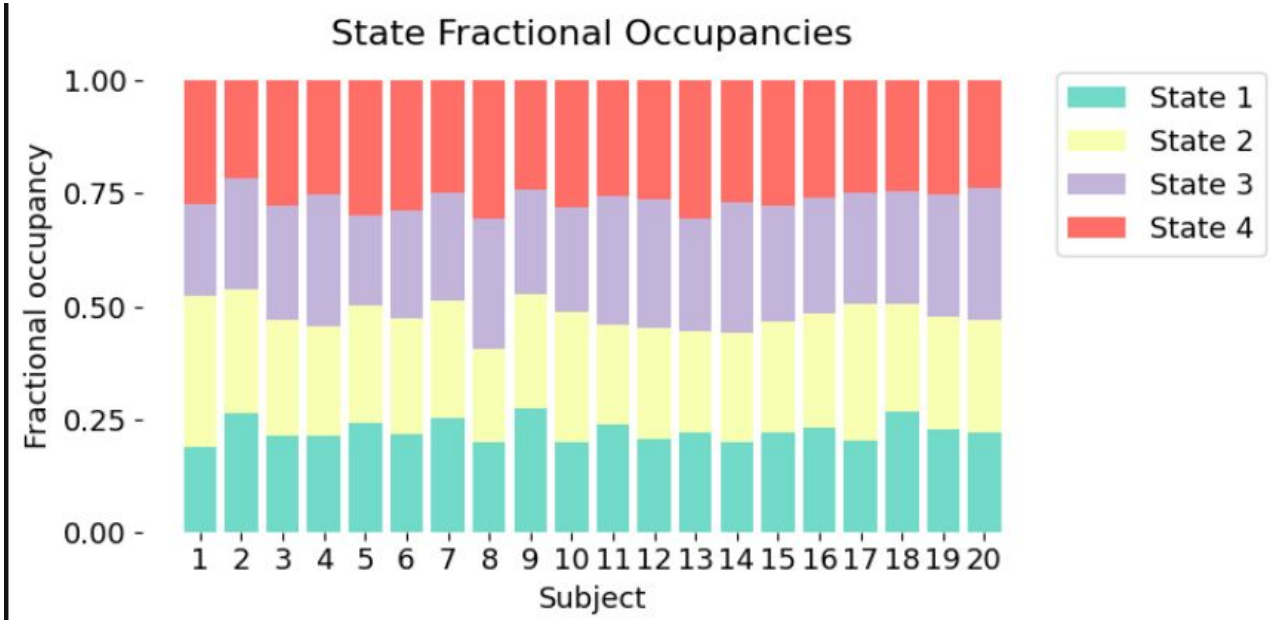
And plot the Fractional Occupancies (see also graphics module):

```
graphics.plot_FO(FO, num_ticks=FO.shape[0])
```

```
FO = utils.get_FO(Gamma, indices=T_t)
```

And plot the Fractional Occupancies (see also graphics module):

```
graphics.plot_FO(FO, num_ticks=FO.shape[0])
```



The state lifetimes (also called dwell times) indicate how long a state is active at a time (either on average, median, or maximum). This can be informative to understand whether states tend to last longer or shorter times, pointing towards slower vs. faster dynamics:

```
LTmean, LTmed, LTmax = utils.get_life_times(vpath, T_t)
```

Now we will plot the mean the state lifetime (see also graphics module):

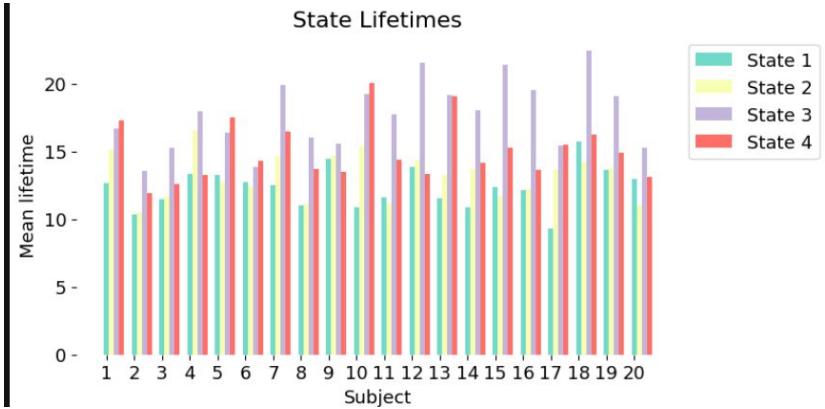
```
graphics.plot_state_lifetimes(LTmean, num_ticks=LTmean.shape[0], ylabel='Mean lifetime')
```

The state lifetimes (also called dwell times) indicate how long a state is active at a time (either on average, median, or maximum). This can be informative to understand whether states tend to last longer or shorter times, pointing towards slower vs. faster dynamics:

```
LTmean, LTmed, LTmax = utils.get_life_times(vpath, T_t)
```

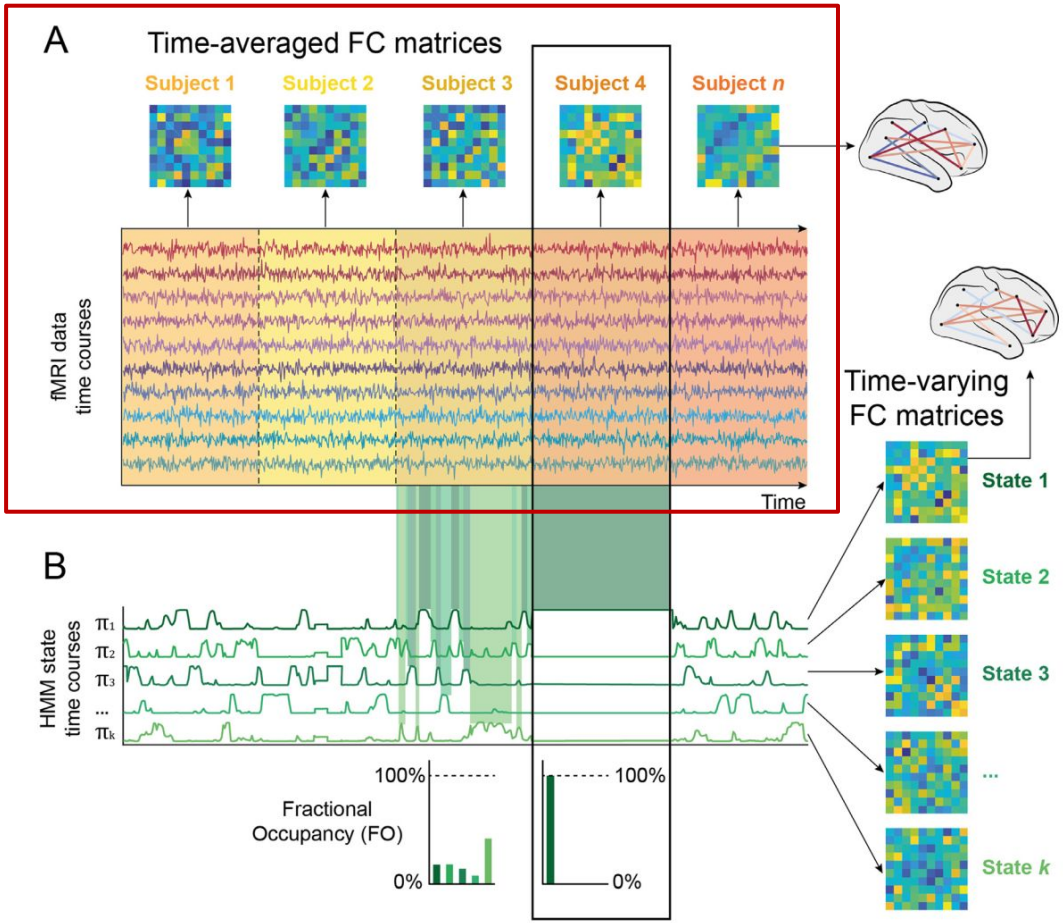
Now we will plot the mean the state lifetime (see also graphics module):

```
graphics.plot_state_lifetimes(LTmean, num_ticks=LTmean.shape[0], ylabel='Mean lifetime')
```



How to think in the model and parameters overall

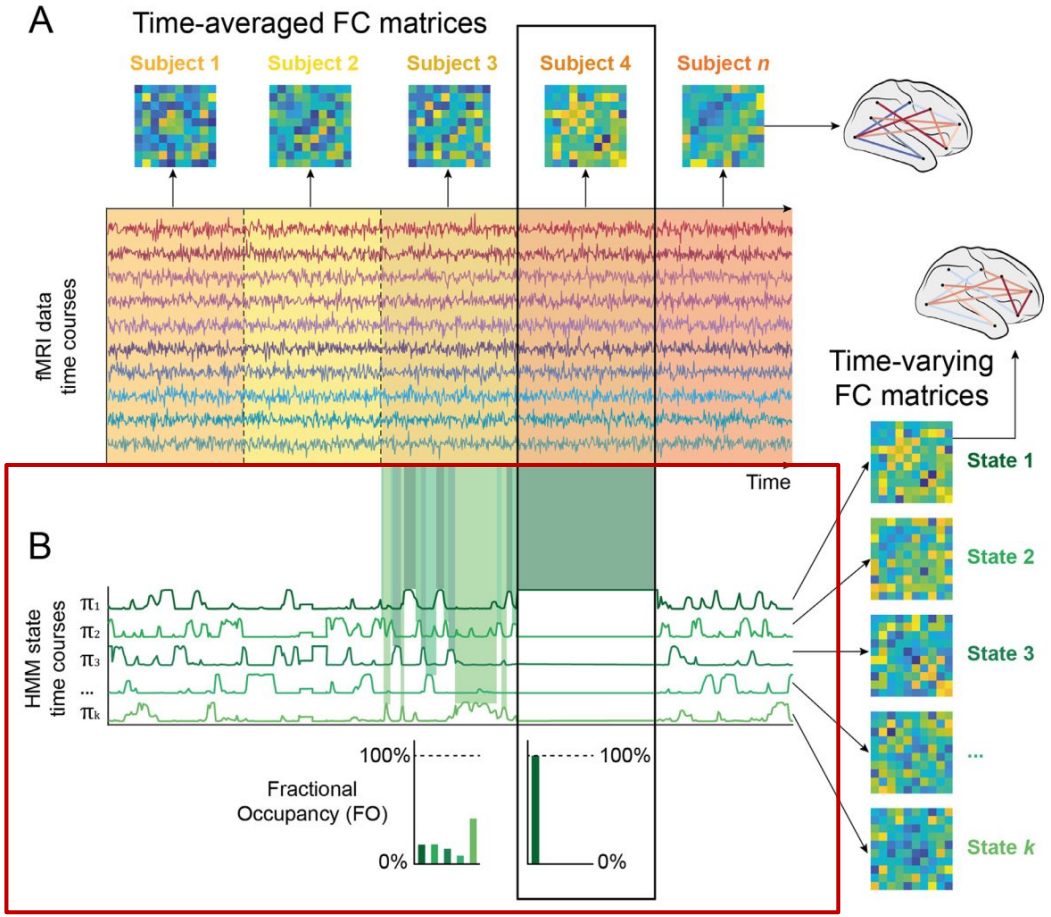
Between-subject variability in time-averaged FC may affect stasis in a time-varying FC model.



https://doi.org/10.1016/j.neuroimage.2022.119026

(A) Time-averaged FC (N by N) matrices for each subject
were obtained by pairwise correlating time courses of all parcels from each subject. Subjects are represented in the time series as different colors. As observed, the
time-averaged FC matrix from Subject 4 is very different from the time-averaged FC matrices of the other subjects.

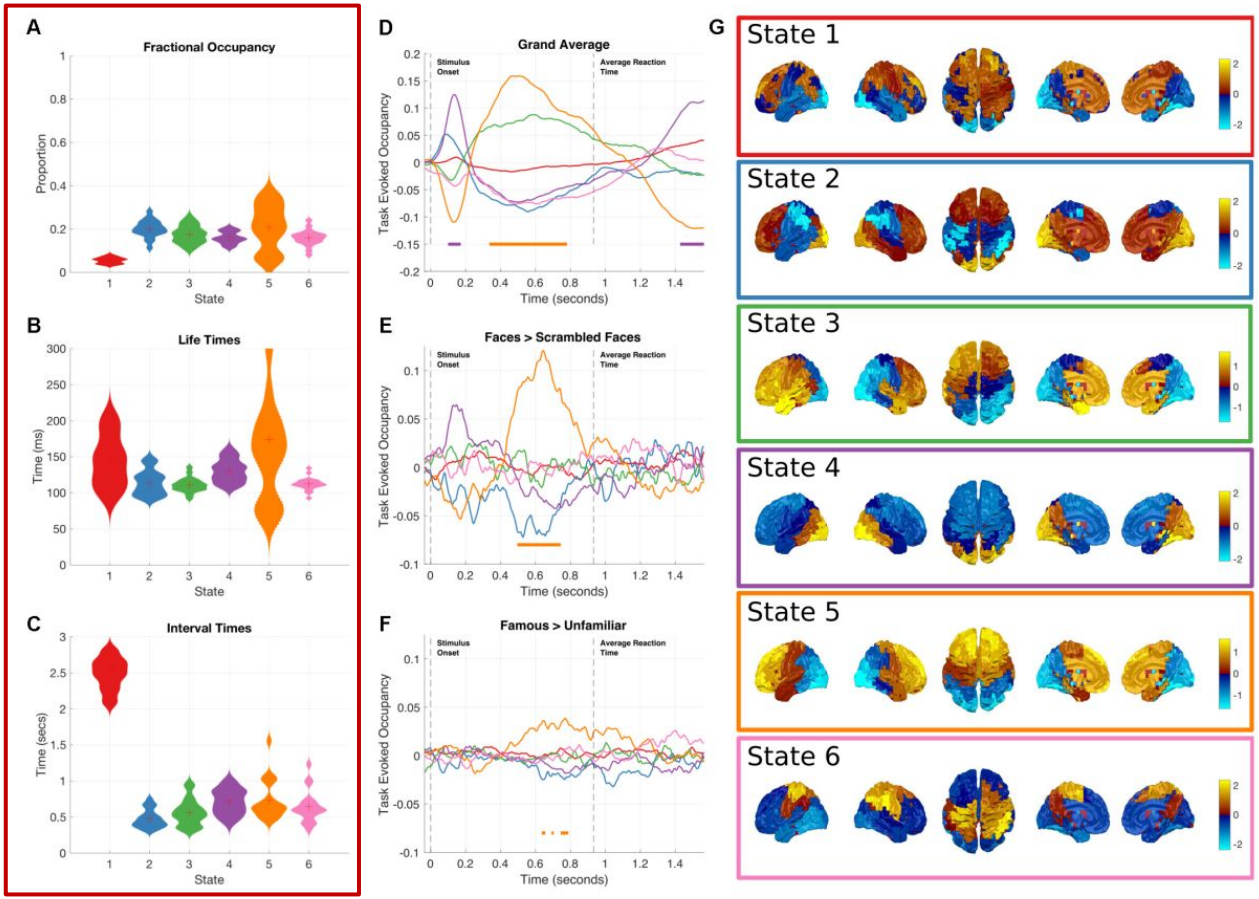
Between-subject variability in time-averaged FC may affect stasis in a time-varying FC model.



https://doi.org/10.1016/j.neuroimage.2022.119026

(B) Given a prespecified number of states K , the Hidden Markov Model (HMM) estimates both the state-specific FC matrices and when the states become active. In the example for Subject 3, all states transiently occur and recur over time. In opposition to this temporal recurrence in Subject 3, the HMM time course for the time points corresponding to Subject 4 stays stable at a high probability for state 1. The temporal recurrence of states can be measured by their fractional occupancy (FO), indicating the proportion of the entire time series that a given state occupies. In this example, state FOs for Subject 3 indicate that all states take up a similar amount of the time series with certain states being relatively more prevalent than others. In Subject 4, however, the FO of state 1 is at 100% while all others are at 0%, since state 1 occupies the entire time series of this subject. This is summarised by the term “stasis”: The model is static when one state’s FO approaches 100% and all others are close to 0%.

Results summary for the Time-Delay Embedded HMM



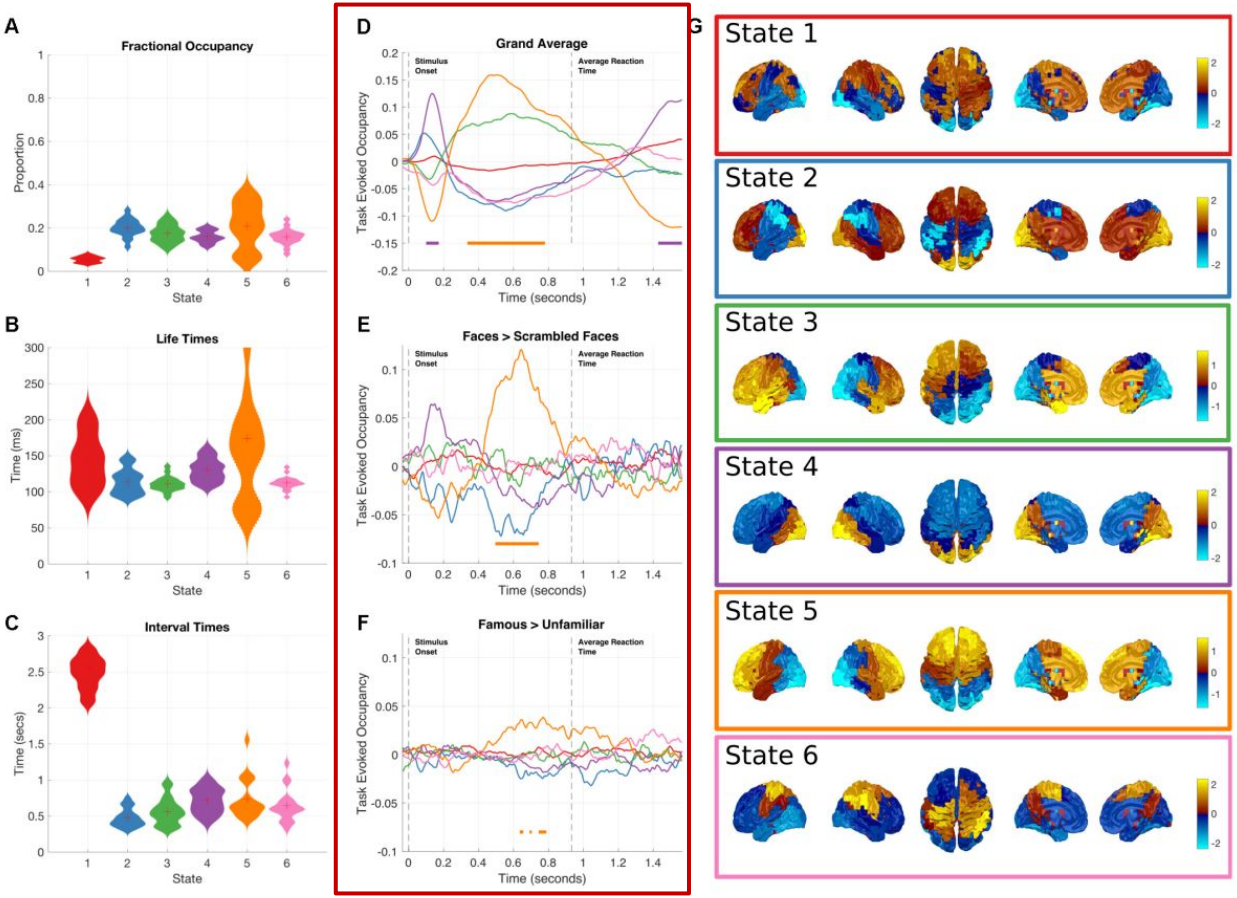
https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2018.00603/full

Results summary for the Time-Delay Embedded HMM, Note that these results are independently estimated from the results in Figure.

The right column shows the overall temporal statistics estimated from the continuous data without considering task structure.
The fractional occupancy (A), Lifetimes (B) and Interval times (C) are shown. The middle column shows the group level results of the GLM analysis computed from the task-evoked fractional occupancies.
(D) shows the mean change in occupancy across all trials relative to baseline. Periods of significant change are indicated by a solid line at the bottom of the plot
color-coded to state. (E) the result of the differential contrast between the Face and Scrambled Face stimuli. (F) the results of the differential contrast between the Famous and Unfamiliar face stimuli.

(G) The mean activation maps and Coherence networks for the six states extracted from the post hoc multi taper estimation. These results reflect wideband activation in each state and are z-scored across parcels.

Results summary for the Time-Delay Embedded HMM



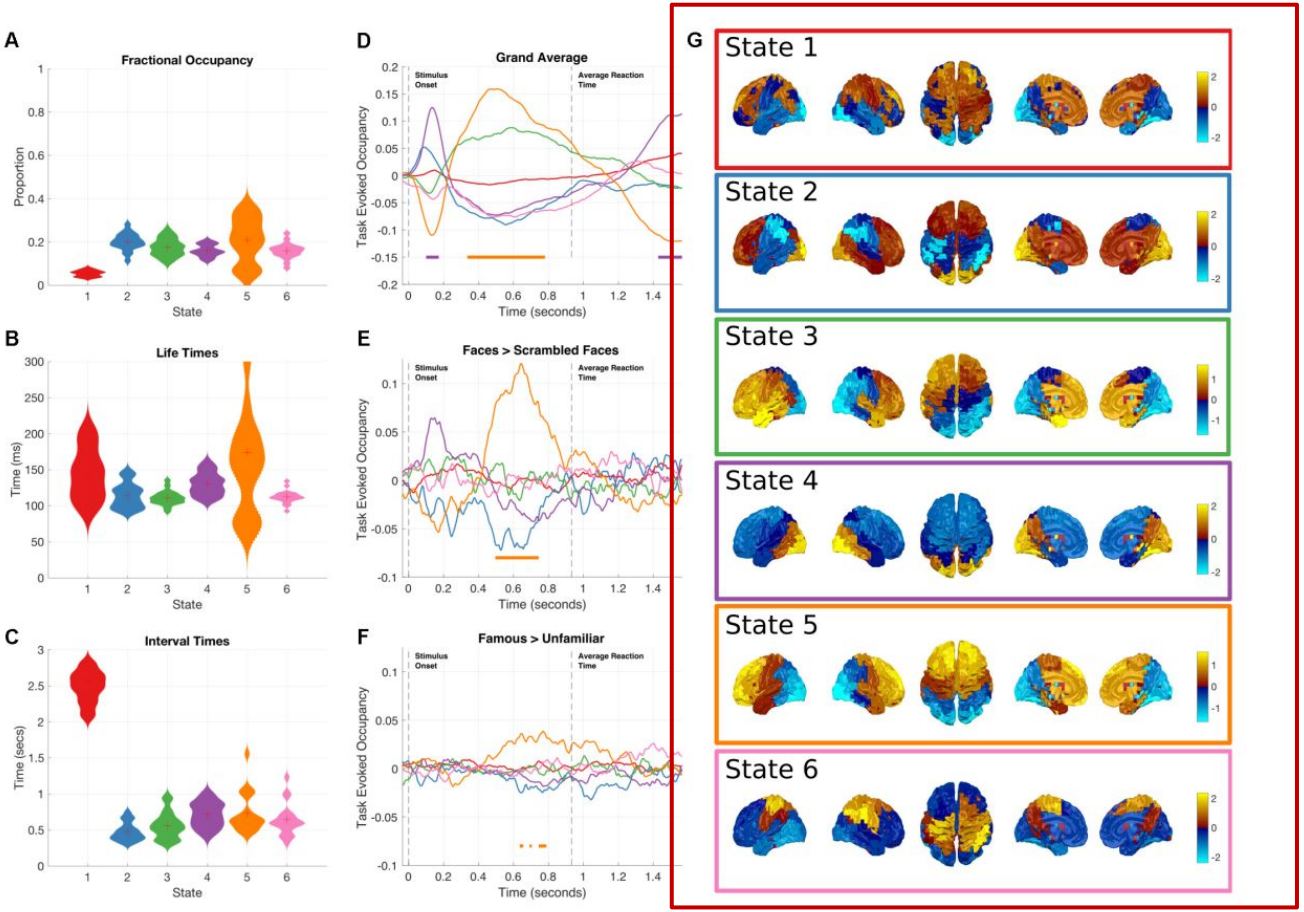
https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2018.00603/full

Results summary for the Time-Delay Embedded HMM, Note that these results are independently estimated from the results in Figure.

The right column shows the overall temporal statistics estimated from the continuous data without considering task structure.
The fractional occupancy (A), Lifetimes (B) and Interval times (C) are shown. The middle column shows the group level results of the GLM analysis computed from the task-evoked fractional occupancies.
(D) shows the mean change in occupancy across all trials relative to baseline. Periods of significant change are indicated by a solid line at the bottom of the plot color-coded to state. (E) the result of the differential contrast between the Face and Scrambled Face stimuli. (F) the results of the differential contrast between the Famous and Unfamiliar face stimuli.

(G) The mean activation maps and Coherence networks for the six states extracted from the post hoc multi taper estimation. These results reflect wideband activation in each state and are z-scored across parcels.

Results summary for the Time-Delay Embedded HMM



https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2018.00603/full

Results summary for the Time-Delay Embedded HMM, Note that these results are independently estimated from the results in Figure.

The right column shows the overall temporal statistics estimated from the continuous data without considering task structure.
The fractional occupancy (A), Lifetimes (B) and Interval times (C) are shown. The middle column shows the group level results of the GLM analysis computed from the task-evoked fractional occupancies.
(D) shows the mean change in occupancy across all trials relative to baseline. Periods of significant change are indicated by a solid line at the bottom of the plot
color-coded to state. (E) the result of the differential contrast between the Face and Scrambled Face stimuli. (F) the results of the differential contrast between the Famous and Unfamiliar face stimuli.

(G) The mean activation maps and Coherence networks for the six states extracted from the post hoc multi taper estimation. These results reflect wideband activation in each state and are z-scored across parcels.

glhmm toolbox can be used for different applications:

glhmm
GAUSSIAN LINEAR HIDDEN MARKOV MODELS

Predicting individual traits from an HMM:
https://glhmm.readthedocs.io/en/latest/notebooks/Prediction_example.html

Across-Subject Testing:
https://glhmm.readthedocs.io/en/latest/notebooks/Testing_across_subjects.html

Overview of Hidden Markov Models (HMMs)

Take home!

- Unsupervised Learning: HMMs are utilized as an unsupervised method to handle time series data.
- Markov Property: They assume that the future state depends only on the current state and not on the sequence of events that preceded it.
- Hidden States: HMMs estimate the hidden states of the data, assigning each time point to a state probabilistically.
- Applications: Commonly used in brain data analysis to capture dynamic brain states that are not directly observable through behavior.

Overview of Hidden Markov Models (HMMs)

Take home!

- Unsupervised Learning: HMMs are utilized as an unsupervised method to handle time series data.
- Markov Property: They assume that the future state depends only on the current state and not on the sequence of events that preceded it.
- Hidden States: HMMs estimate the hidden states of the data, assigning each time point to a state probabilistically.
- Applications: Commonly used in brain data analysis to capture dynamic brain states that are not directly observable through behavior.

Overview of Hidden Markov Models (HMMs)

Take home!

- Unsupervised Learning: HMMs are utilized as an unsupervised method to handle time series data.
- Markov Property: They assume that the future state depends only on the current state and not on the sequence of events that preceded it.
- Hidden States: HMMs estimate the hidden states of the data, assigning each time point to a state probabilistically.
- Applications: Commonly used in brain data analysis to capture dynamic brain states that are not directly observable through behavior.

Overview of Hidden Markov Models (HMMs)

Take home!

- Unsupervised Learning: HMMs are utilized as an unsupervised method to handle time series data.
- Markov Property: They assume that the future state depends only on the current state and not on the sequence of events that preceded it.
- Hidden States: HMMs estimate the hidden states of the data, assigning each time point to a state probabilistically.
- Applications: Commonly used in brain data analysis to capture dynamic brain states that are not directly observable through behavior.

Overview of Hidden Markov Models (HMMs)

Take home!

- Unsupervised Learning: HMMs are utilized as an unsupervised method to handle time series data.
- Markov Property: They assume that the future state depends only on the current state and not on the sequence of events that preceded it.
- Hidden States: HMMs estimate the hidden states of the data, assigning each time point to a state probabilistically.
- Applications: Commonly used in brain data analysis to capture dynamic brain states that are not directly observable through behavior.

Practical Implementation with GLHMM Package

Steps to Implement:

- Define Parameters: Assume the number of states (K) and the probability distribution of these states (observation model).
- State Definitions: Get the definition and probability of each data point belonging to a state.
- Transition Probabilities: Calculate the probability of switching from one state to another.
- Brain Data Dynamics: This approach helps in understanding the dynamics of brain data, revealing how long each state lasts and how quickly subjects switch between states.

Practical Implementation with GLHMM Package

Steps to Implement:

- Define Parameters: Assume the number of states (K) and the probability distribution of these states (observation model).
- State Definitions: Get the definition and probability of each data point belonging to a state.
- Transition Probabilities: Calculate the probability of switching from one state to another.
- Brain Data Dynamics: This approach helps in understanding the dynamics of brain data, revealing how long each state lasts and how quickly subjects switch between states.

Practical Implementation with GLHMM Package

Steps to Implement:

- Define Parameters: Assume the number of states (K) and the probability distribution of these states (observation model).
- State Definitions: Get the definition and probability of each data point belonging to a state.
- Transition Probabilities: Calculate the probability of switching from one state to another.
- Brain Data Dynamics: This approach helps in understanding the dynamics of brain data, revealing how long each state lasts and how quickly subjects switch between states.

Practical Implementation with GLHMM Package

Steps to Implement:

- Define Parameters: Assume the number of states (K) and the probability distribution of these states (observation model).
- State Definitions: Get the definition and probability of each data point belonging to a state.
- Transition Probabilities: Calculate the probability of switching from one state to another.
- Brain Data Dynamics: This approach helps in understanding the dynamics of brain data, revealing how long each state lasts and how quickly subjects switch between states.

Thanks!

More about our work in brain data analysis:

-See our poster !!!

More about the use of the model in different works:

- The Gaussian distribution, on fMRI and other neuroimaging modalities, where the states capture spatial information, i.e. the average activation pattern and functional connectivity (covariance) of the BOLD signal between areas across the whole brain (Vidaurre et al., 2017; Stevner et al., 2019; Vidaurre, et al., 2018).
- A Wishart distribution on fMRI, if we wish to focus on changes in functional connectivity (i.e. covariance; Vidaurre et al., 2021; Ahrends et al., 2022).
- The time-delay embedded distribution on electrophysiological data (e.g. MEG or EEG), which, based on the Gaussian distribution, can capture spectral modulations on highdimensional data (Vidaurre et al., 2018).
- The autoregressive model, also used on electrophysiological data, which, with a larger number of parameters than the time-delay embedded model, offers a finer description of the spectral aspects in the data, and is therefore more appropriate for temporally richer modalities like local field potentials (LFP; Vidaurre et al., 2016; Masaracchia et al., 2023).
- A decoding model to describe the changing relation between brain activity and ongoing stimuli by explicitly including task information in the model. Specifically, this is a regression model where the brain data act as independent variables, and the stimuli as dependent variables (Vidaurre et al., 2019).
- An encoding model, which also describes the relation between brain activity and stimuli with a focus on spatial interpretation. Here, the brain data are the dependent variables, and the stimuli are the independent variables (Higgins et al., 2022).

Notebook in colab:

https://colab.research.google.com/drive/15zorjbmeMyvFidmjPYM1behdXO8yO18L#scrollTo=isxHZiIr79hz

Extra:

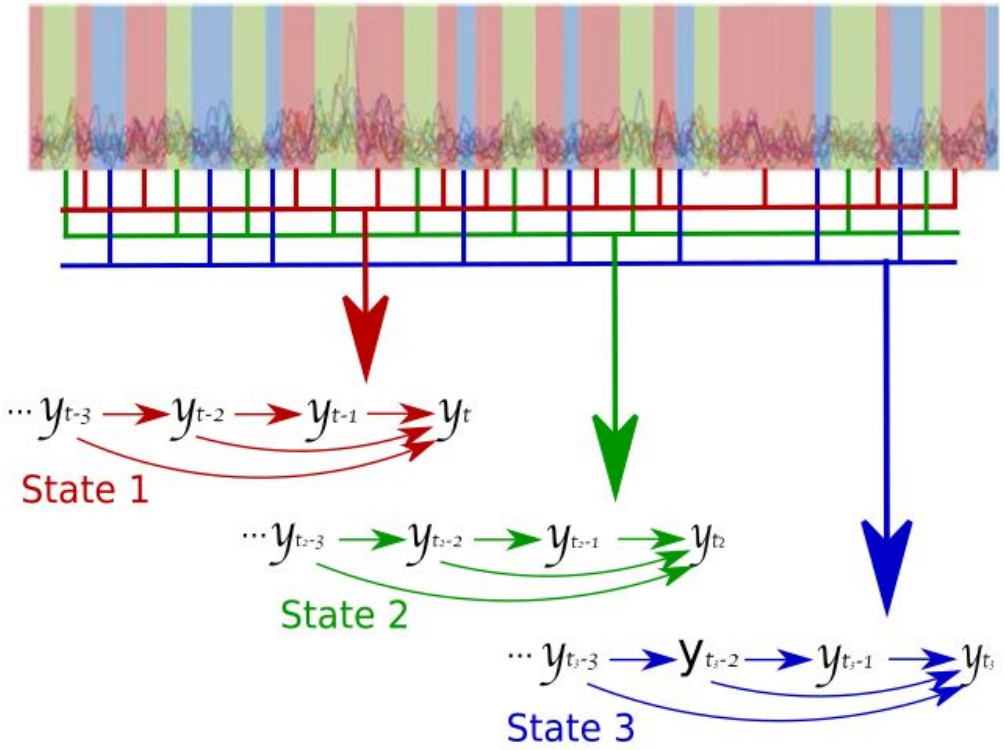
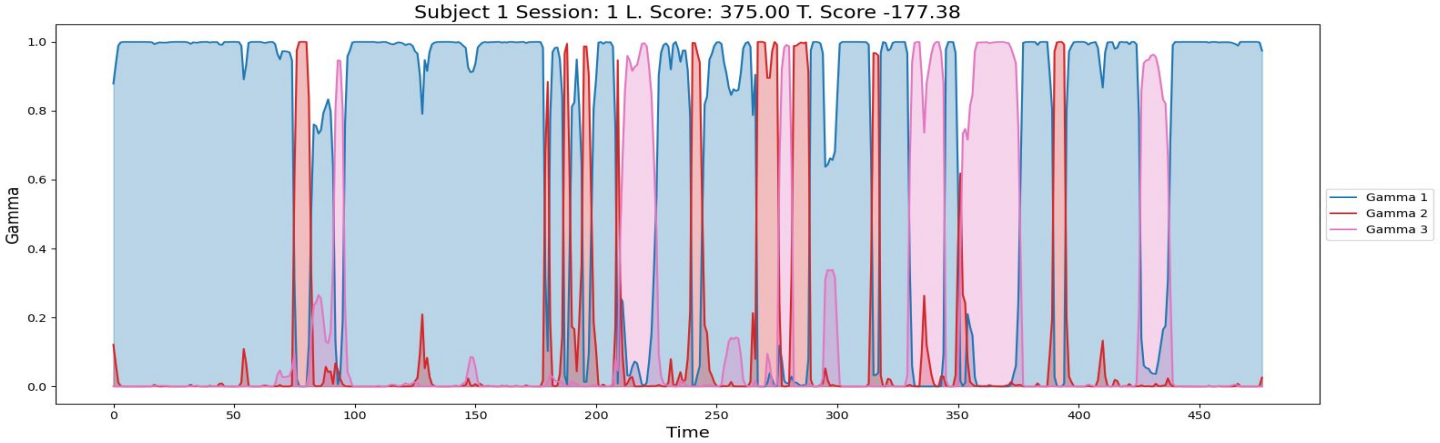


Fig. 2. Graphical representation of the HMM–MAR. The time series (background) is partitioned into three states denoted by the blue, red and green slabs. Each state is characterised by a different set of dynamics, determined by the linear historical interactions between data points $\boldsymbol{y}_t$ (small arrows).

https://github.com/OHBA-analysis/HMM-MAR/wiki/Theory#model-description

- HMM dynamical model is characterized by abstract states.
- Such states are defined at group level.
- States are represented with a Covariance matrix each one (based on parcellation/brain region).
- Each individual visit one of the states at the time (mutually exclusive).



- We have some parameters that characterize each individual dynamics in each state: Fraccion occupancy (FO), swching rate (SW) and lifetimes (Parameters per individual per state)