Supporting Material

An introduction to infinite HMMs for single molecule data analysis

I. Sgouralis and S. Pressé

Contents

S1 Sampler's implementation

Here we present a fully functioning implementation of the iHMM sampler outlined in the Methods section of the main text. This implementation can be used to analyze experimental time series by generating chains of samples as demonstrated in the Results section of the main text and Section S5 below. Herewith, we refer to these chains of samples simply as "chains".

The present implementation can handle only a *single observation time series*. The supplied time series needs to be *corrected for drift* beforehand.

The sampler's source code is developed within the commercial software **MATLAB**® which needs to be installed before the sampler can be used. For an introduction to **MATLAB**, we refer to the user's manual published by **MathWorks, Inc.**

To facilitate the uninitiated user, we provide also a *graphical user interface (GUI)*. The reader who is interested in source code details might skip the sections below and move directly to Sect. S1.3.

S1.1 Download and installation

The code can be downloaded in a single zip file, **sampler.zip**. Once downloaded, **sampler.zip** needs to be unzipped and the contents placed in the same folder. If properly extracted, this folder should include the following: (i) **sampler_SRC**, a subfolder with the sampler's source files; (ii) **sampler_GUI.fig** and **sampler_GUI.m**, the sampler's GUI files; (iii) **demo_data.txt**, a text file with an example time series; and (iv) **demo_sampler.m**, a script demonstrating the use of the sampler's source.

For the activation and use of the GUI see Sect. S1.2. For direct use of the source code see Sect. S1.3.

S1.2 Graphical user interface

To activate the GUI: (i) start **MATLAB**, (ii) set the current folder to the folder that the sampler's files are stored, and (iii) enter **sampler_GUI** in the *command window* or double click on **sampler_GUI.fig** in the *current folder window*.

Screenshots of the GUI are shown on Fig. S1. Through the GUI you can load data (Import), configure the iHMM to be used (Parameters), generate and expand chains (Sample), perform a preliminary analysis (Analyze), and save the chains for further analysis (Export).

S1.2.1 Description

Below we provide a description of the various choices in the GUI. For reference see Fig. S2.

(A) Import button Use this button to import the time series to be analyzed. For the supported formats see Sect. S1.2.2 below.

(B) Imported time series Panel B1 shows the imported time series. Panel B2 shows the distribution of the imported time series.

(C) Clear button Use this button to clear the imported time series and any generated chain.

(D) Parameter values Use these fields to enter values for the iHMM parameters. These are: D1, concentration for the transition probabilities α; D2, concentration for the base γ; D3, hyperparameters for the emission gaussian means and precisions $\lambda, \rho, \beta, \omega$ (for details on the emission model see Sect. S3); D4, initial number of states in the chain $K^{(1)}$. All values must be strictly positive. The parameter λ can take any real value, while ρ, β, ω can take only positive real values. The parameter $K^{(1)}$ can take positive integer values.

Remark: larger/lower values of α tend to generate chains with sparser/denser transition probability vectors, larger/lower values of γ tend to generate chains with more/fewer states. The value of $K^{(1)}$ does not affect the

generated chains provided a sufficient number of samples are generated; nevertheless, $K^{(1)}$ might increase or reduce the burn-in period.

(E) Create button Use this button to create a new chain with the parameter values specified in the fields D. Newly created chains contain only one sample and need to be expanded (see I below) before they can be analyzed (see K below) or exported (see L below).

(F) Batch size Use this field to enter the number of new samples to be generated. If the chain has been just created, this is the number of initial samples to be generated, otherwise the sampler will expand the existing chain by the specified number of samples. This number can be a non-negative integer.

Remark: Generally, the larger the number of requested samples, the longer the computations last.

(G) Burn-in fraction Use this field to set the fraction of total samples considered as burn-in. For example a fraction of 0.20 will discard the first 20% of the samples in the generated chain from subsequent analysis. This fraction needs to be between 0 and 1.

Remark: Burn-in samples are highlighted in J1 and J3 (see below).

(H) Progress report Activate these options to monitor the sampler's progress. Following the completion of each iteration, the status bar option prints messages in the command line, while the visual option uses separate figures to illustrate the last generated sample.

Remark: Activating the option for visual output slows down the computations and should be avoided for the generation of long chains.

(I) Sample button Use this button to start the sampler. The sampler will generate and augment the existing chain by the batch size specified in F.

(J) Sampled chain Shows selected sample sequences from the current chain. Specifically, these are: Panel J1, number of visited states $K^{(r)}$; Panel J2, distribution of $K^{(r)}$; Panel, J4 emission means $\mu^{(r)}$; Panel J4, distribution of $\mu^{(r)}$.

Remark: Burn-in samples (determined according to the value in G) are highlighted in J1 and J3, but are excluded from J2 and J4.

(K) Analyze button Use this button to perform a preliminary analysis of the generated chain. The analysis includes: (i) estimation of the emission means time series, (ii) estimation of the transition probabilities, (iii) estimation of the dwell times. These analyses utilize a discretization of the parameter space which should be provided by the user. The generated figures can be exported using the standard figure options. The results can also be saved upon request. In any case, for a thorough analysis, the chain should be exported (see L below) and analyzed externally.

Remark: Burn-in samples (determined according to the value in G) are excluded from the analyses.

(L) Export button Use this button to export the current chain for further analysis independently of the sampler's GUI. For the supported formats see Sect. S1.2.2 below.

Remark: Burn-in samples (determined according to the value in G) are not included in the exported chains.

(M) Reset button Use this button to delete the current chain and enable the generation of a new one. Reseting does not affect the imported time series in (A)–(B).

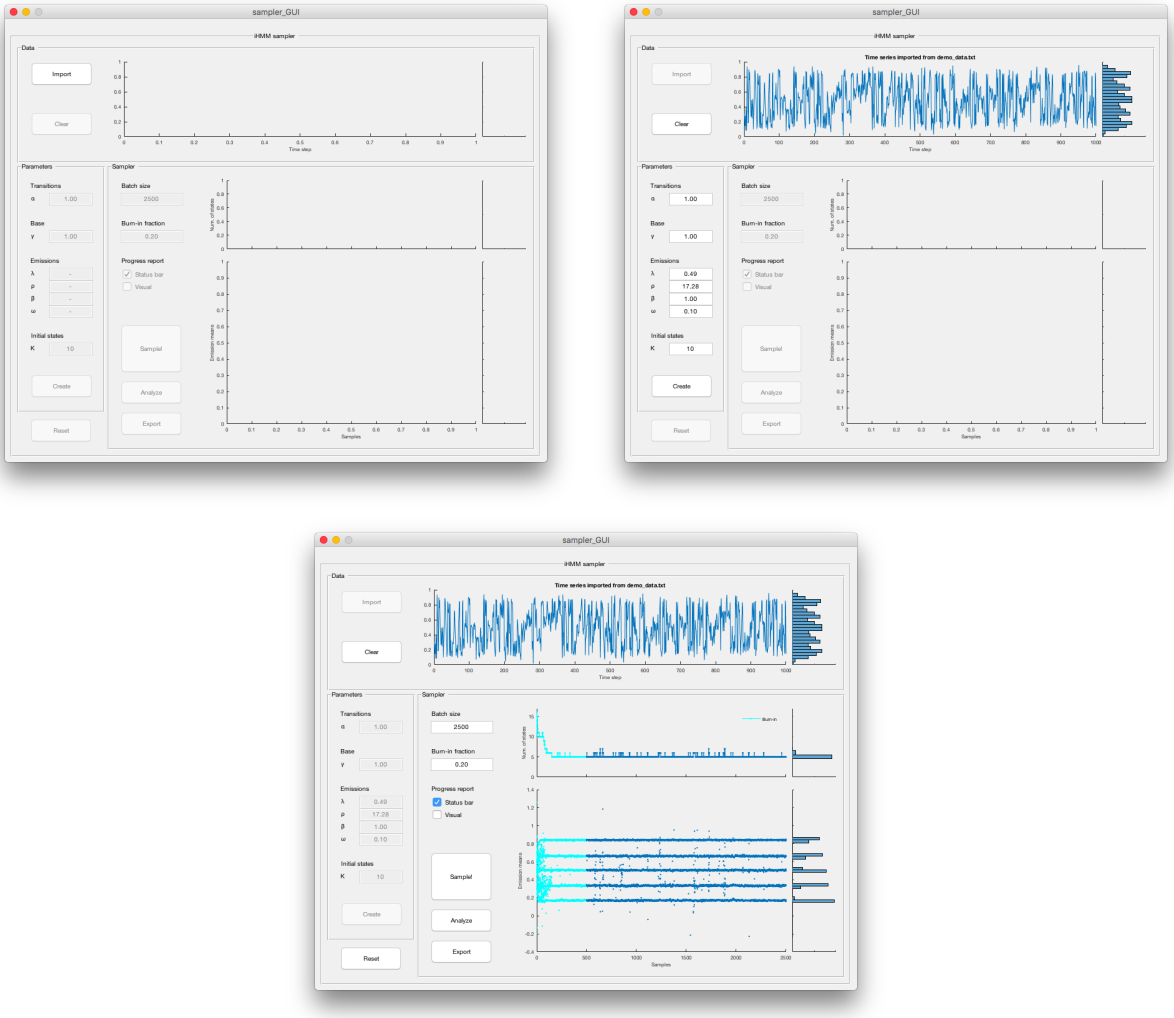

Figure S1: **Graphical user interface.** *Upper left*, without imported data; *Upper right*, with imported data without generated chain; *Lower*, with imported data and generated chain. For details see Sect. S1.2

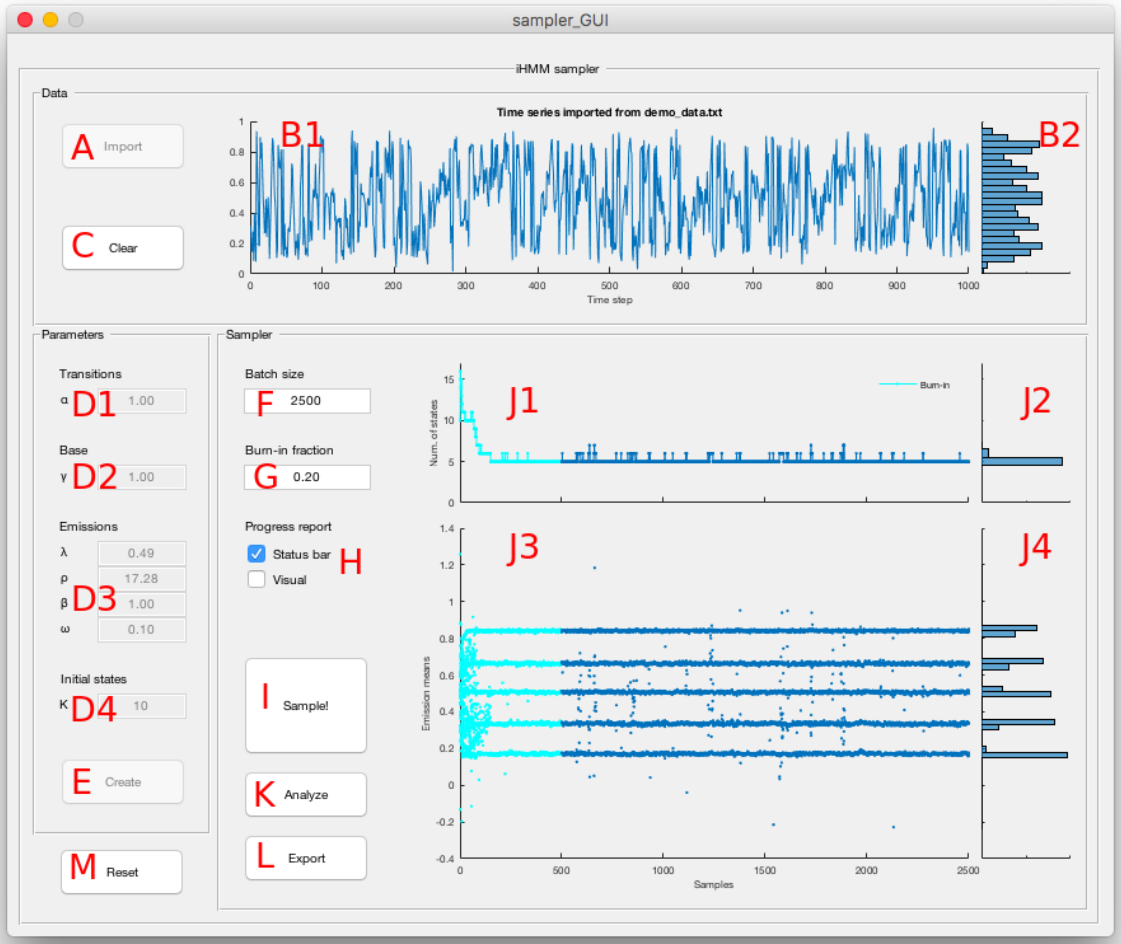

Figure S2: **Graphical user interface.** For details see Sect. S1.2.1

S1.2.2 Supported data formats

The GUI can import experimental time series, denoted $\bar{x}$ in the main text, to be used by the sampler in the generation of the chains, and also export chains to be analyzed by external software packages. Below we describe the data formats that are supported for import/export.

Import The GUI requires time series $\bar{x}$ to be imported in either of the following formats:

- Text files: a single text file (`*.txt`) consisting of one column with one observation per line.
- Mat files: a single mat file (`*.mat`) containing a vector with one observation per component.

Export The sampler can export the generated samples in the following format:

- Mat files: a single mat file (`*.mat`) that contains the following individual chain fields: `params`, `s`, `K`, `B`, `O`, `P`, `F`. These fields are explained in Sect. S1.3.1 below.

S1.3 Source code

The sampler is implemented in the files of the subfolder `sampler_SRC`. These include various `MATLAB` functions that: (i) sample random variables with specified distributions, (ii) carry out the steps of the algorithm briefly described in the methods section of the main text or in detail in Sect. S2 below, (iii) initialize each chain and their parameters, and (iv) provide further handing (i.e. such as export, visualization, and preliminary analysis) of each chain.

The source code can be used directly to analyze a time series without invoking the GUI. For an illustration we refer to `demo_sampler.m`. Here we describe the basic functionally. For a description of the implemented algorithm see Sect. S2 below.

S1.3.1 Chain structure

The code analyzes a time series which is stored in `x`. Throughout the code `x` is a vector of length `N`. Each generated chain, stored in `chain`, is a structure that contains the following fields:

- `chain.params` a structure keeping the chain's parameters. Namely, these are:
  - `params.a` the transitions concentration α.
  - `params.g` the base concentration γ.
  - `params.Q` the hyperparameters of the emission distributions $\lambda, \rho, \beta, \omega$ (for details see Sect. S3) stored in `params.Q(1:4)`, respectively.
  - `params.Brep` and `params.Frep` implementation parameters with no direct interpretation.
- `chain.r_max` is the total number of samples kept in `chain`. Generally, `chain.r_max` takes values larger or equal to 1.
- `chain.dr_sk` is the stride of the `chain`. That is, `chain` stores samples every `chain.dr_sk` iterations. Generally, `chain.dr_sk` takes values greater of equal to 1.
- `chain.i` is the total number of iterations run since the initialization of `chain`. Generally, `chain.i` takes values greater of equal to 1 and lower than `chain.r_max`.
- `chain.s` is an array of size `[chain.r_max,N]`. Each row `chain.s(r,:)` stores the state's label sequence of the `r` sample. Specifically, `chain.s(r,n)` is the label of the state the system attains at the `n` time step. Generally, the labels in `chain.s(r,:)` attain all values in `1:chain.K(r)`.

- `chain.K` is a vector of size `chain.r_max`. Specifically, `chain.K(r)` is the total number of distinct states visited in the `r` sample. Generally, `chain.K(r)` takes integer values above or equal to 1.

- `chain.B` is a cell of size `[chain.r_max,1]`. Each component `chain.B{r}` is a vector of size `chain.K(r)+1` that stores the base of the `r` sample. Specifically, `chain.B{r}(k)` is the component of the base measure corresponding to the states in `chain.s(r,:)` with label `k`, and `chain.B{r}(end)` is the remainder such that `chain.B{r}(:)` sums to 1.

- `chain.O` is a cell of size `[chain.r_max,1]`. Each component `chain.O{r}` is a vector of size `chain.K(r)+1` that stores the initial state probabilities of the `r` sample. Specifically, `chain.O{r}(k)` is the component of the initial probability measure corresponding to the states in `chain.s(r,:)` with label `k`, and `chain.O{r}(end)` is the remainder such that `chain.O{r}(:)` sums to 1.

- `chain.P` is a cell of size `[chain.r_max,1]`. Each component `chain.P{r}` is an array of size `[chain.K(r), chain.K(r)+1]` that stores the transition probability vectors of the `r` sample. Specifically, `chain.P{r}(k,j)` stores the transition probability corresponding to departing from the state in `chain.s(r,:)` with label `k` and arriving to the state in `chain.s(r,:)` with label `j`, and `chain.P{r}(k,end)` are the remainders such that `chain.P{r}(k,:)` sum to 1 for each `k`.

- `chain.F` is a cell of size `[chain.r_max,1]`. Each component, `chain.F{r}` is an array of size `[chain.K(r), 3]` that stores the parameters of the emission distributions of the `r` sample. Specifically, `[chain.F{r}(k,1), chain.F{r}(k,2)]` stores the mean μ and precision τ, respectively, of the emission distribution (see Sect. S3) corresponding to the state in `chain.s(r,:)` with label `k`, while `chain.Fr(k,3)` stores the relative fraction of visits to the states in `chain.s(r,:)` with label `k`.

S1.3.2 Chain handling

The driver function is `chainer_main.m` which creates and expands a single chain. The driver can be called two-fold

- `chain = chainer_main(x,r_max,[],opts,flag_sta,flag_vis)` creates a chain with parameters specified in `opts` and runs `r_max` iterations of the sampler generating `r_max` samples. If `r_max` is 1, the driver simply initializes a chain without generating any samples.

- `chain = chainer_main(x,r_max,chain_init,[],flag_sta,flag_vis)` expands an existing chain, namely `chain_init`, by generating and adding `r_max` new samples.

In both calls, `x` is the experimental time series and `flag_sta` and `flag_vis` are logical flags indicating whether the sampler should display output monitoring each iteration. In particular, activating `flag_sta` prints in the command line (output messages are produced by `chainer_main.m`), while activating `flag_vis` uses separate figure windows (visualization of the sampler's progress is handled by `chainer_visualize.m`). We note that activating `flag_vis` can significantly slow down the sampler, so it should be avoided in the generation of long chains.

In `chainer_main.m`, parameters for new chains are initialized by `chainer_init_params.m` and utilize the options passed in `opts`. Specifically, these options provide: (i) `opts.a`, transitions concentration α; (ii) `opts.g`, base concentration γ; and (iii) `opts.Q(1:4)` emission hyperparameters $\lambda, \rho, \beta, \omega$, respectively (see Sect. S3). Starting samples for each chain are initialized by `sampler_init_sample.m` and utilize `opts.K_init`, the initial number of states $K^{(1)}$.

New samples in `chainer_main.m` are produced by iterative calls to `sampler_update.m`. This function uses an old set of samples: `chain.s(r,:)`, `chain.K(r)`, `chain.B{r}`, `chain.O{r}`, `chain.P{r}`, `chain.F{r}` and produces a new one `chain.s(r+1,:)`, `chain.K(r+1)`, `chain.B{r+1}`, `chain.O{r+1}`, `chain.P{r+1}`, `chain.F{r+1}`. In doing do, `sampler_update.m` uses the functions in `sampler_SRC` that perform the various steps of the beam sampler outlined in Sect. 2.2 of the main text and described in more detail in Sect. S2 below.

S1.3.3 Chain analysis

The source code is equipped with functions that perform simple analyses

- `[m_mod,m_red] = chainer_analyze_means(chain,fr,dr,m_min,m_max,m_num,x)` This function uses the samples in `chain` to compute the time series of the best emission mean estimate, i.e. the most likely sequence $\mu_1 \to \mu_2 \to \cdots \to \mu_N$ where μ_n denotes the mean value of the emission distribution producing the observation x_n at the n^{th} time step. On output `m_mod` is a vector of the same dimensions as the experimental time series `x` with contains the corresponding emission mean and `m_red` is a vector containing only the distinct values in `m_mod`.

- `[m_edges,p_mean,d_dist] = chainer_analyze_transitions(chain,fr,dr,m_min,m_max,m_num,fl)` This function uses the samples in `chain` to compute estimates of the transition probabilities and expected dwell times. These estimates are computes on the basis of a discretization of the parameter space that is returned in `m_edges`. On output, `m_edges` is a vector containing distinct μ values, `p_mean` is a square matrix containing the mean transition probabilities, and `d_dist` is a two column matrix with the first column containing the mean and the second column containing the standard deviation of the expected dwell times. Specifically, `p_mean(k,j)` is the transition probability of departing from the interval between `m_edges(k)` and `m_edges(k+1)` and arriving to the interval between `m_edges(j)` and `m_edges(j+1)`, while `d_dist(k,1:2)` is the mean and standard deviation of the expected dwell time in the interval between `m_edges(k)` and `m_edges(k+1)`. Generally, the transition probabilities result in a sub-stochastic `p_mean` estimate, i.e. row sums may be below 1, since samples in `chain` do not keep track of the unvisited states in each sample.

Both functions determine burn-in in `chain` according to the fraction `fr`. For example, a value for `fr` of 0.20 discards the first 20% of the samples in `chain` as burn-in. From the remaining samples, both functions use only every `dr` sample. That is, a value for `dr` of 2 indicates that for the analyses is used only 1 every 3 samples excluding those in the burn-in period.

Both functions utilize histograms of the corresponding posterior probability distributions. For the creation of these histograms, the parameter space is described between `m_min` and `m_max` using a total of `m_num` equidistant bins. The limits `m_min` and `m_max` are automatically adjusted to include every occurring value in the `chain`.

S1.3.4 Chain utilities

The source code is equipped with the following functions that perform auxiliary tasks:

- `chainer_export(chain,fr,dr,file_name,frmt)` This function exports the samples in `chain` for further analysis with external software. For the supported export formats see Sect. S1.2.2 above. This function produces no output.

- `G=chainer_visualize(G,r,chain,x)` This function is used to provide a visual comparison of the `r` sample in `chain` with the experimental time series in `x`. On input, `G` is a graphic handle that may be empty (in which case a new handle will be initialized). On output, `G` is the updated graphic handle.

S2 Sampler's pseudocode

Below we provide explicit details for the implementation of the sampler's steps. These steps are based on the beam sampler [1] and are briefly described in the Methods section of the main text and implemented in the source code of Sect. S1.3 above. For completeness, we also include an initialization step which is not mentioned in Sect. 2.2 of the main text.

For definitions and notation we refer to the Methods section of the main text. To facilitate bookkeeping, we assume that all states σ_k represented in the statespace $\mathbb{S}$ are labeled with $k = 1, \ldots, K$, while unrepresented states are labeled with $k > K$.

Step 0 Before the sampler takes over, we need to provide values for the first samples $\bar{s}^{(1)}, \tilde{\tilde{\pi}}^{(1)}, \tilde{\beta}^{(1)}$ and $\tilde{\phi}^{(1)}$. These values can be chosen at random, provided they have consistent dimensions.

Then, for $r = 1, 2, \ldots$ we need to iterate the steps that follow.

Step 1 Initially, we need to sample a new sequence $\bar{u}^{(r)}$ of auxiliary (slicers) variables

$$u_n \big| \bar{s}^{(r-1)}, \tilde{\tilde{\pi}}^{(r-1)} \sim \pi^{(r-1)}_{s_{n-1}^{(r-1)} \to s_n^{(r-1)}} \mathcal{U}(0,1) \tag{S1}$$

where $\mathcal{U}(0,1)$ denotes the uniform probability distribution over $[0,1]$.

Step 2 Once $\bar{u}^{(r)}$ is sampled, we need to ensure that all states that the system is allowed to visit are represented in $\mathbb{S}$. The probability of the system departing from a state σ_k in $\mathbb{S}$ and arriving to a state already present in $\mathbb{S}$ equals $\sum_{j=1}^{K} \pi_{\sigma_k \to \sigma_j}$, thus the probability of the system jumping to a state *not* present in $\mathbb{S}$ is $1 - \sum_{j=1}^{K} \pi_{\sigma_k \to \sigma_j}$. Therefore, to test wether we represent all necessary states or not, we have to check against the condition

$$\max_{k=1,\ldots,K} \left(1 - \sum_{j=1}^{K} \pi^{(r-1)}_{\sigma_k \to \sigma_j} \right) < \max_{n=1,\ldots,N} u_n^{(r)} \tag{S2}$$

If the condition fails, we generate a new state σ_{K+1}, add it in $\mathbb{S}$, adjust K, and repeat until the condition is satisfied. Specific details on how to expand $\tilde{\beta}^{(r-1)}$, $\tilde{\tilde{\pi}}^{(r-1)}$, and $\tilde{\phi}^{(r-1)}$, when generating new states, can be found in Ref. [2, 1].

Step 3 At this point, all σ_k that the system may visit are present in $\mathbb{S}$, so we can sample a new state sequence $\bar{s}^{(r)}$ by a minor modification of the forward-backward algorithm initially developed for the finite HMM [3, 4]. For details see Ref. [1].

Step 4 Following the sampling of $\bar{s}^{(r)}$, some of the states currently present in $\mathbb{S}$ might not have been visited. So, in this step, we remove from $\mathbb{S}$ any states not visited, since they are not needed in future iterations. In other words, we discard from $\tilde{\tilde{\pi}}^{(r-1)}, \tilde{\phi}^{(r-1)}, \tilde{\beta}^{(r-1)}$ those components corresponding to the states that are not present in $\bar{s}^{(r)}$. Additionally, to help bookkeeping in future iterations, we relabel the states that remain such that they occupy the slots $k = 1, 2, \ldots, K$, where K is the number of *distinct* states in $\bar{s}^{(r)}$. To do so, we reorder the components of $\tilde{\tilde{\pi}}^{(r-1)}, \tilde{\phi}^{(r-1)}, \tilde{\beta}^{(r-1)}$ and modify $\bar{s}^{(r)}$ accordingly.

Step 5 We must now sample $\tilde{\beta}^{(r)}$ using the stick-breaking construction. This can be easily achieved by introducing a new set of auxiliary variables the computation of which is based on $\bar{s}^{(r)}$, $\tilde{\beta}^{(r-1)}$ and γ. For details see Ref. [2, 5].

Step 6 Next, we need to sample new transition probabilities $\tilde{\tilde{\pi}}^{(r)}$ based on the current stick $\tilde{\beta}^{(r)}$ and state sequence $\bar{s}^{(r)}$. For this, we let $c^{(r)}_{\sigma_k \to \sigma_j}$ denote the number of transitions from state σ_k to state σ_j that occur in $\bar{s}^{(r)}$. It follows from the properties of the Dirichlet process that for each σ_k we may generate $\tilde{\pi}^{(r)}_{\sigma_k}$ by sampling from a Dirichlet distribution that depends on $c^{(r)}_{\sigma_k \to \sigma_j}$, $\tilde{\beta}^{(r)}$ and α. See for example Ref. [2].

Step 7 Finally, having sampled all other variables, we now need to sample new emission parameters $\tilde{\phi}^{(r)}$ for the states currently present in $\mathbb{S}$. Generating these samples depends on the particular choices of emission distributions $F(x;\phi)$ and the prior of the emission parameters $H(\phi)$. Those choices are dictated by the physics of the system at hand and may differ with different experimental conditions, e.g. FRET, force spectroscopy, etc. Generally, sampling $\tilde{\phi}^{(r)}$ can be done efficiently if $F(x;\phi)$ and $H(\phi)$ are conjugate or semi-conjugate such as in the model described in Sect. S3 (see below), although non-conjugate cases can also be handled [6].

S3 Emission model

In this work the emissions are modeled by gaussian distributions, for example as in Eq. (1) of the main text. For convenience the gaussians are parametrized by mean value μ and precision τ (i.e. $\tau = 1/\sigma^2$, where σ is the gaussian standard deviation). With this parametrization the gaussians densities are

$$F(x;\phi) = \sqrt{\frac{\tau}{2\pi}} \exp\left(-\frac{\tau}{2}(x-\mu)^2\right) \tag{S3}$$

To estimate $\phi = (\mu, \tau)$ we adopt the conditionally conjugate model developed in [7]

$$\mu\big|\lambda,\rho \sim \mathcal{N}(\lambda, 1/\rho) \tag{S4}$$

$$\tau\big|\beta,\omega \sim \mathcal{G}(\beta, 1/\omega) \tag{S5}$$

where $\mathcal{N}$ and $\mathcal{G}$ denote the normal and gamma probability distributions which have the densities

$$F_{\mathcal{N}}(\mu;\lambda,\rho) = \sqrt{\frac{\rho}{2\pi}} \exp\left(-\frac{\rho}{2}(\mu-\lambda)^2\right) \tag{S6}$$

$$F_{\mathcal{G}}(\tau;\beta,\omega) = \frac{\tau^{\frac{\beta}{2}-1}}{(\frac{2}{\omega\beta})^{\beta/2}\Gamma(\beta/2)} \exp\left(-\frac{\tau\omega\beta}{2}\right) \tag{S7}$$

In the present implementation, the hyperparameters λ (mean of μ), ρ (precision of μ), β (shape of τ) and ω (rate of τ) need to be specified by the user. This can be achieved by either the corresponding fields in the GUI (see Sect. S1.2), or by the values in `opts.Q(1:4)` in the source code (see Sect. S1.3).

S4 Discrete probability distributions

Here we provide the definitions of the discrete probablity disitributions used in this study. For the sake of continiuty, in this section we follow the same notation as in the main text.

S4.1 Categorical probability distributions

Let s be a Categorical random variable

$$s\big|\tilde{\pi} \sim Cat_{\mathbb{S}}(\tilde{\pi}) \tag{S8}$$

where the categorical weights are $\tilde{\pi} = (\pi_{\sigma_1}, \pi_{\sigma_2}, \ldots, \pi_{\sigma_L})$ and the state space is $\mathbb{S} = \{\sigma_1, \sigma_2, \ldots, \sigma_L\}$, i.e. the values s may attain. It is understood that the individual weights π_{σ_k} are non-negative and sum to 1. The probablity distribution of s is givem by

$$\mathbb{P}(s = \sigma_k\big|\tilde{\pi}) = \pi_{\sigma_k} \tag{S9}$$

which may also be equivallently denoted as

$$\mathbb{P}(s\big|\tilde{\pi}) = \prod_{k=1}^{L} (\pi_{\sigma_k})^{\delta_{\sigma_k}(s)} \tag{S10}$$

where $\delta_{\sigma_k}(\cdot)$ denotes the Dirac delta function supported on σ_k, i.e. $\delta_{\sigma_k}(s) = 1$ if $\sigma_k = s$ or $\delta_{\sigma_k}(s) = 0$ if $s \neq \sigma_k$.

S4.2 Dirichlet probability distribution

Let $\tilde{\pi}$ be a (symmetric) Dirichlet random variable

$$\tilde{\pi} \sim Dir_{\mathbb{S}}\left(\frac{\alpha}{L}, \frac{\alpha}{L}, \ldots, \frac{\alpha}{L}\right) \tag{S11}$$

where it is understood that the concentration α is positive. The probability density of $\tilde{\pi}$ is given by

$$\mathbb{P}(\tilde{\pi}) = \frac{\Gamma(\alpha)}{\Gamma^L(\alpha/L)} \prod_{k=1}^{L} (\pi_{\sigma_k})^{\frac{\alpha}{L}-1} \tag{S12}$$

where $\Gamma(\cdot)$ denotes the Gamma function.

S4.3 Posterior probability disitribution for the Dirichlet-Categorical model

Let s and $\tilde{\pi}$ follow a Categorical-Dirichlet model

$$s\big|\tilde{\pi} \sim Cat_{\mathbb{S}}(\tilde{\pi}) \tag{S13}$$

$$\tilde{\pi} \sim Dir_{\mathbb{S}}\left(\frac{\alpha}{L}, \frac{\alpha}{L}, \ldots, \frac{\alpha}{L}\right) \tag{S14}$$

Then the posterior probability density of the categorical weights is given by

$$\mathbb{P}(\tilde{\pi}\big|s) \propto \mathbb{P}(s\big|\tilde{\pi})\mathbb{P}(\tilde{\pi}) = \prod_{k=1}^{L} (\pi_{\sigma_k})^{\frac{\alpha}{L}-1+\delta_{\sigma_k}(s)} \tag{S15}$$

S5 Example data analyses

In this section we show additional analyses of the two datasets mentioned in the Results section of the main text. For the generation of these datasets see Section S5.2 (below). Both sets are analyzed assuming concentrations $\alpha = 1$, $\gamma = 1$ and the following values for the hyperparameters: $\lambda = 0.5$, $\rho = 11.11$, $\beta = 1$, and $\omega = 0.0025$, for the definitions of these see Sect. S3 (above).

S5.1 Kinetics estimation

Figures S3 and S4 show the estimated transition matrices for these datasets. As can be seen, iHMM provides accurate estimates for the transition probabilities within the portions of the state space that are visited in the corresponding traces. For example, the lower two states σ_1 and σ_2 in dataset 2 are accurately estimated using either the reduced trace or the full trace. In contrast, the upper three states σ_3, σ_4, and σ_5 are estimated only from the full dataset. Notice that, because of the fewer visits to σ_5, the accuracy of the corresponding estimates is severely reduced.

S5.2 Data generation

For the generation of the data used in the examples of the Results section of the main text, we simulated Markov models that switch between 5 different states that are denoted $\sigma_1, \ldots, \sigma_5$. The kinetics of these models are shown in Fig. S5. Initial states, denoted σ_0, are chosen from the corresponding stationary distributions.

Each simulated σ_k emits gaussian observations. For simplicity, we assume that the gaussian means are equidistant and that the gaussian standard deviations are identical and equal to 30% of the inter-mean distance.

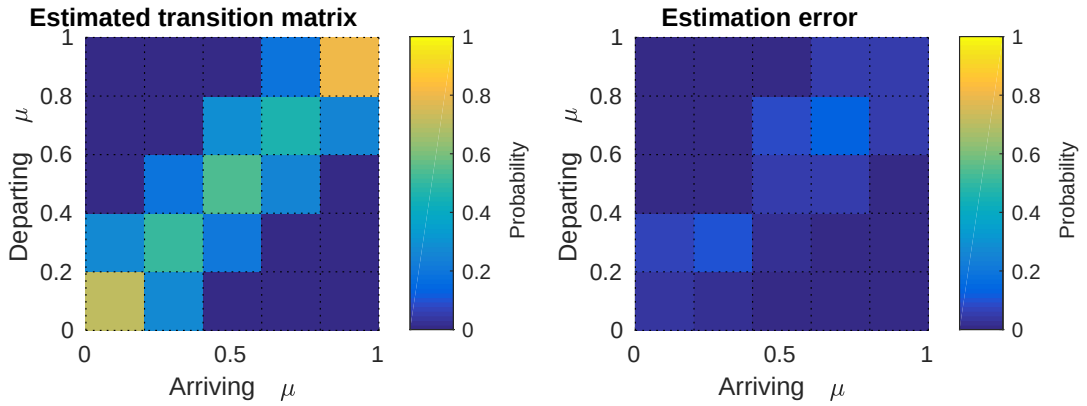

Figure S3: **Kinetics estimation.** Estimated transition matrix (left) and corresponding errors (right) from *dataset 1* of the main text. For details see Sect. S5.

References

[1] J Van Gael, Y Saatci, YW Teh, and Z Ghahramani. Beam sampling for the infinite hidden Markov model. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1088–1095. ACM, 2008.

[2] YW Teh, MI Jordan, MJ Beal, and DM Blei. Hierarchical Dirichlet processes. *Journal of the American Statistical Association*, 2012.

[3] L Rabiner and B Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, 3(1):4–16, 1986.

[4] SL Scott. Bayesian methods for hidden Markov models. *Journal of the American Statistical Association*, 2011.

[5] EB Fox, EB Sudderth, MI Jordan, and AS Willsky. A sticky HDP-HMM with application to speaker diarization. *The Annals of Applied Statistics*, pages 1020–1056, 2011.

[6] RM Neal. Markov chain sampling methods for Dirichlet process mixture models. *Journal of Computational and Graphical Statistics*, 9(2):249–265, 2000.

[7] D Görür and CE Rasmussen. Dirichlet process gaussian mixture models: Choice of the base distribution. *Journal of Computer Science and Technology*, 25(4):653–664, 2010.

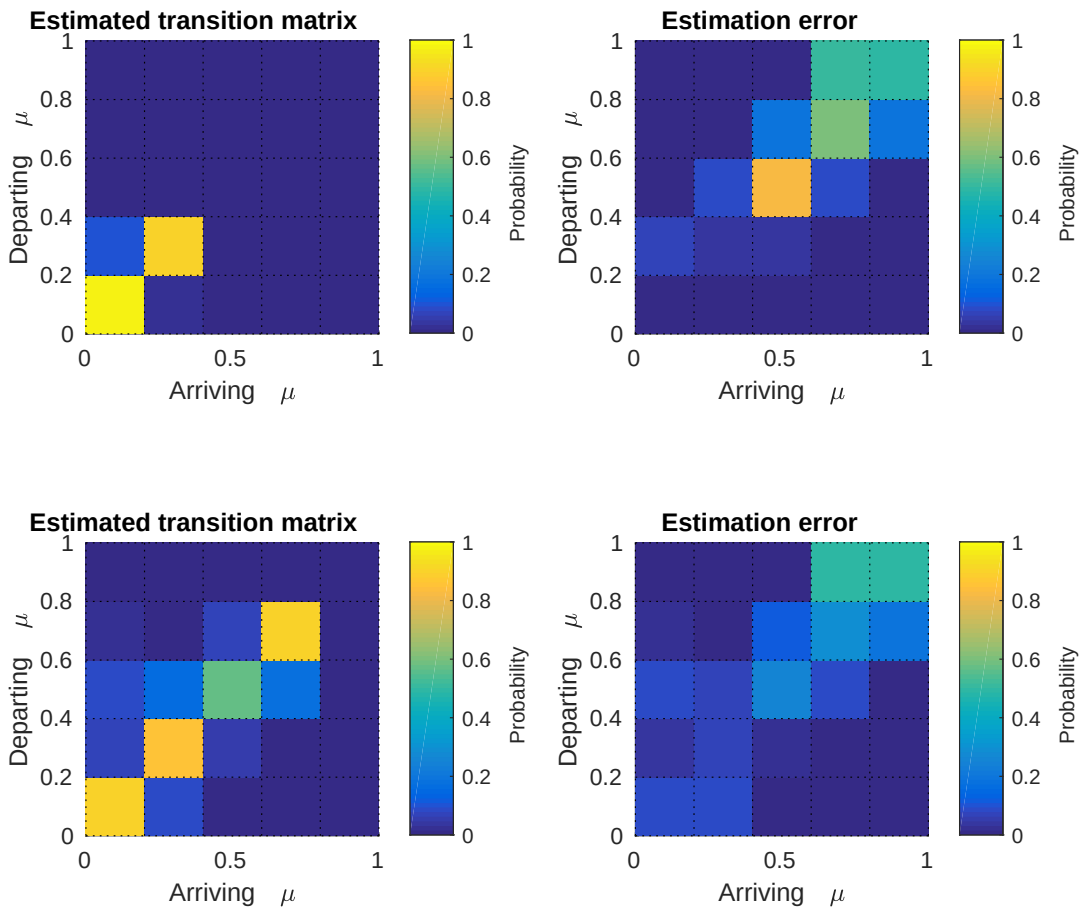

Figure S4: **Kinetics estimation.** Estimated transition matrix (left) and corresponding errors (right) from *dataset 2* of the main text. Upper panels correspond to the first part the dataset; while lower panels correspond to the full dataset. For details see Sect. S5.

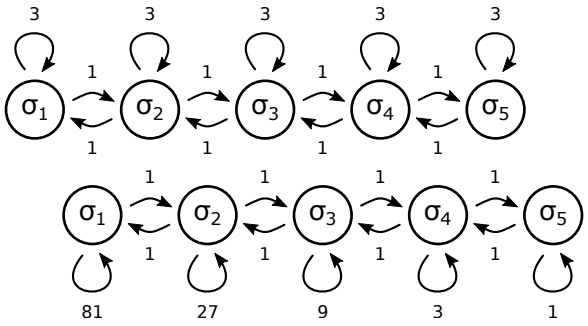

Figure S5: **Markov models used to generate the synthetic data in the Results section of the main text.** *Upper* model generated dataset 1, and *lower* model generated dataset 2. For details see main text and Sect. S5.