vbSPT (Variational Bayes for Single Particle Tracking) – User-guide and Documentation

Fredrik Persson*, Martin Linden*, Cecilia Unoson, and Johan Elf

May 31, 2013

vbSPT is a program suite for analysis of single particle diffusion trajectories, where the diffusion constants switch randomly according to a discrete Markov process. The program runs on Matlab, but uses compiled C-code to speed up the most computer intensive loops. The latest version of the software as well as a forum for discussion and questions can be found at `sourceforge.net/projects/vbspt/`.

If you use it for academic research, please cite the original article in your work:

Fredrik Persson, Martin Lindén, Cecilia Unoson, and Johan Elf, *Extracting intracellular diffusive states and transition rates from single molecule tracking data*, Nature Methods 10(3):265269, 2013. doi:10.1038/nmeth.2367:

*freddie.persson@gmail.com ; bmelinden@gmail.com

Contents

1 vbSPT software user manual

In this section, we introduce the developed vbSPT software and how to use it.

1.1 Recommended hardware

It is computationally very demanding to identify the model that describes thousands of trajectories the best. For this reason it is strongly recommended that the analysis of data sets with more than 1000 trajectories are run on a very good computer. For example, a typical analysis with 10000-15000 trajectories including bootstrapping takes 2-10 h on a 2 x Intel Xeon X5650 (6 core, 2.66 GHz, 12MB L3) machine running MATLAB verR2012a. It should be noted that this version is capable of working with 12 parallel nodes, while previous versions only work with 8, setting a limit on the actual number of usable cores in the computer. However, the test example (see Sec. 1.3) should produce a result in less than 10 min also on a conventional desktop or laptop computer.

1.2 Installation

To install the vbSPT software, uncompress the vbSPT.zip file into a dedicated folder, which we will call `vbRoot/` in these notes. The files under `vbRoot/HMMcore/` might need to be recompiled, depending on what system you are running. To do this, make sure you have a C compiler installed on you system and go to the folder `vbRoot/HMMcore/` and run the script `compile_code`. For a list of compatible compilers see:
http://www.mathworks.se/support/compilers/.

It is recommended to add the `vbRoot/` folder and the subfolders `Tools`, `HMMcore/`, and `VB3/` to the Matlab path. This can be done by executing the matlab script `vbSPTstart` in the folder `vbRoot/`. If it is not added to the Matlab path it has to be called from the same folder or with its full path. Instructions for how to add these folders permanently to your Matlab path can be found in the Matlab documentation.

It is also recommended to start using a static folder structure for the analysis to keep it simple and benefit from the use of relative paths (used in the GUI). The proposed structure consist of two subfolders (*e.g.* `InputData/` and `Results/`) located in a folder containing the runinputfiles that defines the analysis parameters.

1.3 Test runs

In order to test if the installation is correct and working we have included a small set of sample data and the corresponding runinputfiles in the folder `vbRoot/ExampleData/`.

The example data set consists of 500 trajectories, with lengths chosen from am exponential distribution with an average trajectory length of 10 positions, and a minimum of two positions. The data set was generated by the script `ExampleData/InputData/inputScript_example.m`, with the following parameters:

Parameter	**Example data**
timestep [s]	0.003
P_1	0.67
P_2	0.33
D_1 [μm^2 s^{-1}]	1.0
D_2 [μm^2 s^{-1}]	3.0
A_{12} [timestep^{-1}]	0.042
A_{21} [timestep^{-1}]	0.084

To test if the software is working follow these steps:

- Add the above mentioned VB3 folders to your Matlab path, *e.g.* by executing `vbSPTstart`.
- Navigate Matlab to the `vbRoot/ExampleData/` folder.
- Execute the command `R=VB3_HMManalysis('runinput_short.m')` in the Matlab prompt.

This should produce a file called `testresult_vbSPT_HMM_short.mat` in the `vbRoot/Results/` subfolder, and also return the result in the matlab struct `R`. If it does not work, disable parallel computing by setting `parallelize_config=false` in the runinputfile and rerun the analysis. For a first look at the results, execute the following:
`VB3_getResult('runinput_short.m')`. Note that the diffusion constant is here given in units from the input data, so in this case [$nm^2 s^{-1}$]. A graphical representation can be invoked by `VB3_displayHMMmodel('runinput_short.m')`. Here the diffusion constants are shown in [μm^2 s^{-1}], provided that the length and time units are given as in the GUI (see Section 1.6).

It should be noted that the `runinput_short.m` runinputfile specifies an analysis of the data that actually ignores a large part of the trajectories, by having a minimum trajectory length of 7. A larger analysis, that could take 20 min to 1 h on a laptop, can be started by running `R=VB3_HMManalysis('runinput_normal.m')`. However, due to the small amount of data in the example data sets, one cannot expect splendid numerical agreement with the input parameters.

1.4 Analysis input

The analysis takes two kinds of input:

- **runinputfile** - The file containing the parameters defining the input data as well as the analysis, see Section 1.5.
- **trajectories** - A .mat file containing at least one variable that is a cell array where each element, representing a trajectory, is a matrix where the rows define the coordinates in one, two or three dimensions in subsequent timesteps. The number of dimensions to be used for the analysis will be set by the runinputfile.

The analysis is started either from the GUI or by the command `VB3_HMManalysis('runinputfilename')` in the Matlab prompt.

1.5 The runinput file

At the center of the analysis is the runinputfile where the starting parameters are set. This file also acts as a handle for accessing the results and input data and can be used to do *e.g.* extra bootstrapping analysis using the scripts presented in Section 1.8.

The runinputfile can be altered and modified by hand just as a text file or generated and edited through the graphical user interface (GUI). The parameters required in a runinputfile are listed and explained in Table 1.

1.6 The graphical user interface (GUI)

The GUI is started from the Matlab prompt by the command `vbSPTgui`. It should be noted that within the GUI runinputfiles are referred to as scripts. From within the GUI it is possible to create new runinputfiles/scripts, load and edit as well as run them. It is also possible to print the result from a previous analysis in the Matlab prompt by loading its runinput file and choosing 'Show Result'. The GUI limits the input data to be in length units of either [nm] or [μm] and time units to [s]. This is for the convenience of being able to supply initial guesses and results for diffusion coefficients in the common unit [$\mu m^2\ s^{-1}$]. If other units are desired the user is limited to manually modifying the runinputfiles.

1.7 Analysis results

Here, we list the Matlab notation for some important variables contained within the result given by the analysis code. The analysis saves the result in a .mat file containing the following variables:

- **INF** - An array that documents the progress of the search algorithm, one converged model per row. Each row contains the **I**teration number (restart during which the model was generated), the **N**umber of states, and the lower bound on the evidence, **F**, which is the model score.
- **Wbest** - A structure describing the best global model found by the analysis.
- **WbestN** - A cell array containing structures that describes the best model for each model size as found by the analysis.
- **bootstrap** - A structure containing the bootstrapping result for the best global model and also the best model for each model size provided that 'fullBootstrap=true' was given in the runinput file (or chosen in the GUI).
- **dF** - An array showing the relative difference in the model score for different model sizes. The size for the best global model should have value 0.
- **options** - A structure with fields defined by the runinput file that is used to run the analysis.

In Table 2 some important fields in the Wbest, and thus also WbestN$\{i\}$ (where i is a number describing the model size) structure are presented and explained. All state-related variables are sorted after increasing diffusion coefficient. In Table 3 some important fields in the bootstrap structure are presented and explained.

1.8 Useful scripts

Here follows a brief description of some included useful scripts. For further information on the input arguments and the scripts please refer to the documentation in the .m files either by opening them or running `help scriptname` from the Matlab prompt.

- `VB3_getResult` - Loads the results from a previous analysis and prints some parameters in the Matlab prompt.
- `VB3_readData` - Loads the trajectory data set used for an analysis.
- `VB3_varyData` - Converges the best models for different model sizes with increasing amounts of input data.
- `VB3_generateSynthData` - Generates trajectories in a *E. coli* like geometry (tubular with spherical endcaps), according to model and geometry parameters specified by the user. Alternatively, the script used

for generating the example data set (`inputScript_example.m`, in the folder `vbRoot/ExampleData/InputData/`) can be modified and used to provide the input for this function.

- `VB3_bsResult` - Bootstraps the results of a finished analysis. Note that the bootstrapping parameters are taken from the the runinput file or options structure given as input, which should therefore be modified prior to running this script.

Additional undocumented scripts that can serve as templates for visualizing and extracting data can be found under `vbRoot/Tools/` and subfolders therein.

1.9 A note on units

The testdata as well as the real data analyzed in the original paper, measured lengths in nanometers and time in seconds, and hence we divide diffusion constants by 10^6 to convert them to the more convenient units of $\mu m^2\ s^{-1}$. The GUI allows the user some choice in specifying what units to use, and then writes the runinputfile accordingly.

The analysis software in itself does not know about units however, and will analyze data in arbitrary units in a consistent manner. The unit of time is specified in the **timestep** parameter of the runinput file (see next section), while the length unit is set by the data. Hence, the user must make sure that diffusion constants in the runinputfile (for initial guesses and priors) are given in the same unit system.

1.10 More information

For more information about any .m file, run `help filename` in the Matlab prompt, where filename should start with 'VB3_'.

2 Tables of variables

Table 1: Explanations of the parameters defined in the runinputfiles. Int and real denotes integers and real numbers respectively, most of which need to be positive if not state otherwise. The units of some quantities are indicated as [time], [length^2/time], etc, indicating the need to choose units consistent with the data.

Parameter	Value	Description
`inputfile`	'filename'	Name of the .mat file that contains the trajectories.
`trajectoryfield`	'trajfield'	Name of the field in the .mat file that contains the trajectories to be analysed.
`parallelize_config`	true/false	Determines whether parallel computing should be used.
`parallel_start`	'command'	Command used to start the parallelization.
`parallel_end`	'command'	Command used to end the parallelization.
`outputfile`	'filename'	Name of the .mat file where the results are saved.
`jobID`	'description'	Description of the job for your own records.
`timestep`	real	Timestep between points in the trajectories, in units of [time].
`dim`	int	Dimensionality of the data to be analysed (the first dim columns in the coordinate data will be used).
`trjLmin`	int	Minimum length of trajectories to be included in the analysis. Recommended default value: 2.
`runs`	int	Number of analysis attempts at each model size. Recommended to use a multiple of the number of cores when running in parallel.
`maxHidden`	int	Maximum number of hidden states to consider. Recommended to use twice the amount of expected hidden states.

Continued on next page

Table 1 – continued from previous page

Parameter	Value	Description
`maxIter`	int	Maximum number of VB iterations. Set to an empty matrix (`[]`) to use the recommended default value of 1000.
`relTolF`	real	Convergence criterion for the relative change in likelihood lower bound. Recommended to set to `[]`, which uses the default value of 10^{-8}.
`tolPar`	real	Convergence criterion for the M-step parameters. Recommended to set to `[]`, which uses the default value of 10^{-2}.
`stateEstimate`	true/false	Determines whether extra large and computer intensive estimates, including Viterbi paths, should be computed.
`bootstrapNum`	int/0	Number of bootstrap resamplings. Set to 0 to disable bootstrapping. 100 or more resamplings are recommended when using this feature.
`fullBootstrap`	true/false	Determines whether bootstrapping should be done for all model sizes, not only the best global model. Bootstrapping all model sizes requires more computer time, but can give an indication of how robust the model size estimate is.
`init_D`	[real real]	Interval for initial guess of diffusion coefficients, given in unit of [length^2/time].

Continued on next page

Table 1 – continued from previous page

Parameter	Value	Description
`init_tD`	[real real]	Interval for initial guess of mean dwell times (lifetimes) of the hidden states, given in units of [time]. Recommended default value: `[2 20]*timestep`. Guessing too short dwell times does not hurt convergence.
`prior_piStrength`	int	Prior strength for the initial state probability (assumed uniform), given in pseudocounts and should be set low. Recommended default value: 5.
`prior_D`	real	Diffusion coefficient prior mean value, in units of [length^2/time]. An order of magnitude estimate is good enough.
`prior_Dstrength`	int	Strength of the diffusion coefficient prior. Recommended default value: 5.
`prior_tD`	real	Dwell time prior mean value, in units [time]. Must not be smaller than `2*timestep`. Default: `10*timestep`.
`prior_tDstrength`	real	Strength of the transition probability prior. Recommended default value: `2*prior_tD/timestep`

Table 2: Important variables within the Wbest structure. The states in the model is always sorted after increasing diffusion coefficient.

`Wbest.`	**Description**
`dim`	Dimensionality of the analysed data.
`N`	Number of states in the model.
`T`	The trajectrory length for all trajectories used in the analysis.
`F`	The score of the model.
`est.Ptot`	The occupation of each state.
`est.Amean`	The transition matrix. States the probability of going between two states within a timestep, *i.e.* Amean(2, 3) gives the probability that a molecule in state 2 will transition to state 3 within a timestep.
`dwellMean`	The mean dwelltime/lifetime for each state, given in units of timesteps.
`DdtMean`	D*timestep in the same length units as the data put into the analysis.
`est2.sMaxP`	The sequence of most likely hidden states for each trajectory. The est2 field is only computed if `stateEstimate=true;` is given in the runinput file (equivalent of choosing 'Additional estimates' in the GUI) since it is rather large and computer intensive.

Table 3: Important variables within the bootstrap structure.

`bootstrap.`	**Description**
`wbs`	All the individual bootstraps containing score (F), the .est field (see Tab. 2) and the index of the randomly chosen trajectories (ind).
`Wmean.est`	The mean value from all the bootstraps for the .est field from Tab. 2.
`Wstd`	The standard deviation from all the bootstraps for all variabls in the .est field from Tab.2.
`WmeanN`	Same as Wmean but for all model sizes. Computed if `fullBootstrap=true` was given in the runinput file (or chosen in the GUI).
`WstdN`	Same as Wstd but for all model sizes. Computed if `fullBootstrap=true` was given in the runinput file (or chosen in the GUI).
`Fbootstrap`	The score for each bootstrap and model size. Computed if `fullBootstrap=true` was given in the runinput file (or chosen in the GUI).
`pBest`	The fraction of all bootstraps that resulted in the corresponding model sizes, *i.e.* pBest(3)=0.8 means that 80% of the bootstraps resulted in a 3 state model. Computed if `fullBootstrap=true` was given in the runinput file (or chosen in the GUI).

3 Introduction to the variational algorithm for single particle tracking

In this chapter, we describe the statistical method used to analyze the data, with more details than the corresponding part of the supplementary information of the original paper[1].

Our approach is to model the state kinetics by a hidden Markov Model (HMM) for diffusing particles with memory-less jumps in diffusion constants, which we analyze by an approximate approach to a maximum evidence model selection, known as variational Bayes or ensemble learning[2,3]. Variational algorithms for HMMs have been derived earlier[4–8], and applied successfully in a biophysical setting to *e.g.* single molecule FRET data[7,8]. The main advantage of the variational maximum evidence approach over the more common maximum likelihood approach to HMMs is the inherent complexity control, *i.e.* the ability to not only learn parameter values from data, but also to make model selection and learn the number of hidden states[8].

3.1 Model selection by maximum evidence

In this section, we briefly introduce our model selection criteria, maximum evidence. For simplicity, we use x, s, θ, and N to denote tracking data, hidden states corresponding to the data, unknown parameters in the model and the number of hidden states, respectively. For readers new to Bayesian statistics, we also recommend the brief introduction by Eddy[9] (or textbooks[2,3]).

A probabilistic model specifies the probability of obtaining the data x and hidden states s, given a model N and some parameter values θ: $p(x, s|\theta, N)$. Our particular model will be specified in Sec. 3.3 below. To use this for Bayesian model selection, we treat all variables in this function as random variables, use the laws of probability to derive the inverse probability $p(N|x)$, which we interpret as a statement about our degree of confidence in model N given the data x, and prefer the most likely model.

This will require us to specify prior distributions, of the form $p(\theta, N)$, that express our beliefs about the models and their parameters prior to seeing the data x. Indeed, the willingness to assign a probability distribution to unknown parameters and treat them on an equal footing with other random variables is a characteristic of Bayesian statistics.

Returning to the derivation of $p(N|x)$, we start by computing the weighted average over all possible hidden states, known as marginalizing over them (or, in physics jargon, integrating or summing them out):

$$p(x|\theta, N) = \sum_s p(x, s|\theta, N). \tag{S1}$$

Using Bayes' rule on $p(x|\theta, N)$, and introducing the aforementioned prior

distributions, we can derive the joint model and parameter probability as

$$p(\theta, N|x) = \frac{p(x|\theta, N)p(\theta, N)}{p(x)}, \tag{S2}$$

where the denominator is a normalization constant,

$$p(x) = \sum_{N'} \int d\theta' p(x|\theta', N')p(\theta|N')p(N'). \tag{S3}$$

Finally, the sought inverse probability is obtained by marginalizing over the parameters,

$$p(N|x) = \frac{\int d\theta p(x|\theta, N)p(\theta, N)}{p(x)} = \frac{1}{p(x)} \int d\theta \sum_s p(x, s|\theta, N)p(\theta|N)p(N), \tag{S4}$$

where we also re-wrote the joint probability in the form $p(\theta, N) = p(\theta|N)p(N)$.

If we further assume that $p(N)$ is constant, *i.e.* that all models (in some interval $0 < N \leq N_{\max}$) are equally probable *a priori*, and note that the denominator $p(x)$ is independent of parameters and models, then the best model is the one that maximizes the numerator in Eq. (S4), also known as the *evidence*,

$$N_{\mathrm{ME}} = \mathrm{argmax}_N \int d\theta \sum_s p(x, s|\theta, N)p(\theta|N), \tag{S5}$$

where $\mathrm{argmax}_y\ f(y)$ denotes the value of y that maximizes the function f.

3.2 Variational maximum evidence

In practice, the integrals and sums in the evidence are intractable for almost all interesting models, and further progress requires good approximations and computers. The difficulties resemble those of computing partition functions in statistical physics, and the two common fallbacks in that field – Monte Carlo simulations[2,10,11] and mean field theory[2–4] – work here as well. The variational or ensemble learning approach we will follow corresponds to mean field theory, where we seek approximations to the posterior distribution in Eq. (S2) in the form

$$p(s, \theta|x, N) = \frac{p(s, x|\theta, N)p(\theta|N)}{p(x|N)} \approx q(s)q(\theta), \tag{S6}$$

i.e., where the hidden states s and parameter values θ are statistically independent. A general recipe to derive mean-field-like approximations of this type[2] is to rewrite the logarithm of the evidence as the solution of an optimization problem in an arbitrary distribution $q(\theta, s)$ over the unknowns.

The approximation then consists of optimizing in a restricted space, in this case that of separable distributions of the form $q(\theta)q(s)$.

To begin with, we multiply and divide the evidence by $q(\theta, s)$, and then make use of Jensen's inequality to write

$$\begin{aligned}\ln p(x|N) &= \ln \int d\theta \sum_s q(\theta, s) \frac{p(x, s|\theta, N)p(\theta|N)}{q(\theta, s)} \\ &\geq \int d\theta \sum_s q(\theta, s) \ln \frac{p(x, s|\theta, N)p(\theta|N)}{q(\theta, s)} \\ = \int d\theta \sum_s \Big[q(\theta, s) \ln p(x, s|\theta, N)p(\theta|N) &- q(\theta, s) \ln q(\theta, s) \Big] \equiv F[q(\theta, s), x].\end{aligned} \tag{S7}$$

This inequality is true for any distribution $q(\theta, s)$. Note that if we use the calculus of variations to directly optimize F with respect to $q(\theta, s)$ subject to a normalization constraint (since $q(\theta, s)$ is a probability distribution), we find $q^*(\theta, s) = \frac{p(x,s|\theta,N)p(\theta|N)}{p(x|N)}$. Substituting back in Eq. (S7), we then get $F[q^*, x] = p(x|N)$, *i.e.* the original intractable problem.

To make this a useful approximation, we place restrictions on the variational distribution $q(\theta, s)$. Our approximate model selection will be to prefer the model for which the tightest bound F can be found within that restricted function space, and we can also use the corresponding optimal distribution $q^*(\theta, s)$ for approximate inference about the parameter values and hidden states [2,3].

A useful restriction is that of separable distribution $q(\theta, s) \approx q(\theta)q(s)$ [4,6,7], which does give a tractable problem. In particular, optimizing with respect to $q(\theta)$ and $q(s)$, while using Lagrange multipliers to enforce normalization, one can derive the following equations:

$$\ln q(\theta) = -\ln Z_\theta + \ln p(\theta|N) + \langle \ln p(x, s|\theta, N) \rangle_{q(s)}\,, \tag{S8}$$

$$\ln q(s) = -\ln Z_s + \langle \ln p(x, s|\theta, N) \rangle_{q(\theta)}\,, \tag{S9}$$

where Z_θ and Z_s are normalization constants, and $\langle \cdot \rangle_{q(\cdot)}$ denotes an average with respect to $q(\cdot)$. As it turns out, these averages can be computed for our model. Following earlier variational treatments of HMMs [4,6,7], we solve these equations iteratively, by alternatingly updating $q(s)$ and $q(\theta)$ for fixed $q(\theta)$ and $q(s)$, respectively. The resulting algorithm greatly resembles the well-known expectation-maximization procedure for maximum-likelihood optimization in HMMs [5] (see Sec. 4 for details).

3.3 Diffusion model

We assume that we can track the particles in d dimensions, and call $\vec{x}_t$ the position at time t. The time between consecutive measurements is Δt, but

we are going to use t as an integer index as well. We model the position by simple diffusion,

$$\vec{x}_{t+1} = \vec{x}_t + \sqrt{2D_{s_t}\Delta t}\vec{w}_t, \tag{S10}$$

where $\vec{w}_t$ are independent d-dimensional Gaussian variables with uncorrelated components of zero mean and unit variance.

Binding and unbinding events are modeled as jumps in the diffusion constant D_{s_t}, as indicated by the degree of freedom $s_t \in \{1, 2, \ldots N\}$, where N is the number of diffusive states. This degree of freedom constitutes our hidden state, and we model it as a discrete Markov process with a transition matrix $\mathbf{A}$, and initial state probabilities $\vec{\pi}$, *i.e.*

$$p(s_1 = j) = \pi_j, \quad p(s_t = j | s_{t-1} = i) = A_{ij}, \text{ for } t > 1, \tag{S11}$$

where normalization demands $\sum_{k=1}^{N} \pi_k = \sum_{k=1}^{N} A_{jk} = 1$. The number of hidden states N is a parameter to be determined from the data. The vector of diffusion constants corresponding to the different hidden states is denoted $\vec{D} = (D_1, D_2, \ldots, D_N)$. We will also use the shorthand notation $\vec{x}_{1:T} = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_T\}$ and $s_{1:T-1} = \{s_1, s_2, \ldots, s_{T-1}\}$ for the positions and hidden states of a whole trajectory.[1]

For our analysis, we need expressions for the joint distribution of positions, hidden states, and parameters, $p(\vec{x}_{1:T}, s_{1:T-1}, \vec{D}, \mathbf{A}, \vec{\pi}|N)$, which can be factorized as

$$\begin{aligned} p(\vec{x}_{1:T}, s_{1:T-1}, \vec{D}, \mathbf{A}, \vec{\pi}|N) = p(\vec{x}_{1:T}|s_{1:T-1}, \vec{D}) p(s_{1:T-1}|\mathbf{A}, \vec{\pi}) \\ \times p(\vec{D}|N) p(\mathbf{A}|N) p(\vec{\pi}|N). \end{aligned} \tag{S12}$$

The first two factors on the above right hand side are specified by the model. In particular, the distribution of hidden state sequences is

$$p(s_{1:T-1}|\mathbf{A}, \vec{\pi}) = \pi_{s_1} \prod_{t=1}^{T-2} A_{s_t s_{t+1}} = \prod_{m=1}^{N} \pi_m^{\delta_{m,s_1}} \prod_{t=1}^{T-2} \prod_{k,j=1}^{N} A_{kj}^{\delta_{k,s_t}\delta_{j,s_{t+1}}}, \tag{S13}$$

and the distribution of positions, conditional on the hidden states, is given by

$$\begin{aligned} p(\vec{x}_{1:T}|s_{1:T-1}, \vec{D}) &= p(\vec{x}_1) \prod_{t=1}^{T-1} p(\vec{x}_{t+1} - \vec{x}_t | s_t, \vec{D}) \\ &= p(\vec{x}_1) \prod_{t=1}^{T-1} \frac{1}{(4\pi D_{s_t}\Delta t)^{d/2}} e^{-\frac{1}{4D_{s_t}\Delta t}\Delta \vec{x}_t^2}. \end{aligned} \tag{S14}$$

[1] Since s_T does not influence the position data, we exclude it from the model.

In these equations, $\delta_{j,k}$ is the Kronecker delta. The initial position distribution $p(\vec{x}_1)$ does not depend on any parameter of interest to us here, and we will drop it from the analysis. It is also convenient to introduce the inverse diffusion constant $\gamma_j = 1/4D_j\Delta t$, and write

$$\begin{aligned} p(\vec{x}_{1:T}|s_{1:T-1},\vec{\gamma}) &= \prod_{t=1}^{T-1}\left(\frac{\gamma_{s_t}}{\pi}\right)^{d/2} e^{-\gamma_{s_t}(\vec{x}_{t+1}-\vec{x}_t)^2} \\ &= \prod_{t=1}^{T-1}\prod_{k=1}^{N}\left(\frac{\gamma_k}{\pi}\right)^{\frac{d}{2}\delta_{k,s_t}} e^{-\delta_{k,s_t}\gamma_k(\vec{x}_{t+1}-\vec{x}_t)^2}. \end{aligned} \tag{S15}$$

The last three factors in Eq. (S12) are the prior distributions for the parameters, expressing our beliefs about the parameter values for different model sizes (values of N), before seeing the data. For computational convenience, independent priors, *e.g.* $p(\vec{\gamma}|N)p(\mathbf{A}|N)p(\vec{\pi}|N)$, are used instead of the more general $p(\vec{\gamma},\mathbf{A},\vec{\pi}|N)$. For the same reason priors functional forms are chosen as conjugate priors (see Chap.2.4.2 in Bishop[3]). As we will see in the derivation below, this means gamma distributions for the inverse diffusion constant priors, and Dirichlet distributions (a multidimensional version of the beta distribution) for the initial state and transition probability priors. Within these constraints, we strive to choose uninformative, or weak, priors in order to let the data speak for itself as much as possible.

3.4 Treatment of many trajectories

Typical *in vivo* single particle tracking experiments produce many rather short trajectories. Most trajectories contain less than 20 consecutive positions, and probably very few (≤ 2) transitions. If considered in isolation, such trajectories are not very informative, and transitions are difficult to identify accurately. To extract meaningful information, we need to pool many trajectories. The simplest way to do that is to assume that they are statistically independent and governed by the same model and parameter set, *i.e.* that all molecules (including those from different cells in the same batch) are equivalent and that the interaction dynamics do not change over time. In that case, the probability distribution for a set of M trajectories is a product of single-trajectory densities governed by the same model,

$$p(\{\vec{x}^i_{1:T}, s^i_{1:T-1}\}_{i=1}^M|\vec{\gamma},\mathbf{A},\vec{\pi},N) = \prod_{i=1}^{M} p(\vec{x}^i_{1:T}, s^i_{1:T-1}|\vec{\gamma},\mathbf{A},\vec{\pi},N), \tag{S16}$$

where different trajectories are indicated by the index i.

4 Derivation of the variational algorithm

We now derive the core elements of the variational algorithm for single trajectories, following earlier Variational Bayesian/Ensemble learning treatments of HMMs[4,6,8]. To start with, we will feed the model, Eqs. (S12-S15),

$$p(\vec{x}_{1:T}, s_{1:T-1}, \vec{\gamma}, \mathbf{A}, \vec{\pi}|N) = p(\vec{x}_{1:T}|s_{1:T-1}, \vec{\gamma})p(s_{1:T-1}|\mathbf{A}, \vec{\pi})p(\vec{\gamma}|N)p(\mathbf{A}|N)p(\vec{\pi}|N), \quad \text{(S12)}$$

$$p(s_{1:T-1}|\mathbf{A}, \vec{\pi}) = \prod_{m=1}^{N} \pi_m^{\delta_{m,s_1}} \prod_{t=1}^{T-1} \prod_{k,j=1}^{N} A_{kj}^{\delta_{k,s_t}\delta_{j,s_{t+1}}}, \quad \text{(S13)}$$

$$p(\vec{x}_{1:T}|s_{1:T-1}, \vec{\gamma}) = \prod_{t=1}^{T-1} \prod_{k=1}^{N} \left(\frac{\gamma_k}{\pi}\right)^{\frac{d}{2}\delta_{k,s_t}} e^{-\delta_{k,s_t}\gamma_k(\vec{x}_{t+1}-\vec{x}_t)^2}. \quad \text{(S15)}$$

into the mean-field machinery of Eqs. (S8) and (S9) (with $\theta = \vec{\gamma}, \mathbf{A}, \vec{\pi},$),

$$\ln q(\theta) = -\ln Z_\theta + \ln p(\theta|N) + \langle \ln p(x, s|\theta, N) \rangle_{q(s)}\,, \quad \text{(S8)}$$

$$\ln q(s) = -\ln Z_s + \langle \ln p(x, s|\theta, N) \rangle_{q(\theta)}\,. \quad \text{(S9)}$$

which also includes choosing functional forms for the prior distributions that make the averages tractable. In the next subsections, we first derive the parameter distributions and the update rule that constitute the M-step of the EM iterations, then go on to the distribution of hidden states and its update rule (the E-step), and finally describe how to compute the lower bound F after a completed E-step.

4.1 Parameters

We have three types of parameters, $\theta = (\vec{\pi}, \mathbf{A}, \vec{\gamma})$, and it will turn out that if we choose priors that factorize in a clever way,

$$p(\theta|N) = p(\vec{\gamma}, \vec{\pi}, \mathbf{A}|N) = p(\vec{\pi}|N) \prod_j p(\gamma_j|N)p(A_{j,:}|N), \quad \text{(S17)}$$

where $A_{j,:}$ means row j of $\mathbf{A}$, then the variational distributions factorize in the same way, which simplifies the computations. Hence, if we substitute factorized priors of this form, plus Eqs. (S13) and (S15), into the parameter distribution equation, Eq. (S8), we get the following variational distribution

for the parameters:

$$\begin{aligned}\ln q(\vec{\pi},\mathbf{A},\vec{\gamma}) &= -\ln Z_\theta + \ln p(\mathbf{A}|N) + \ln p(\vec{\pi}|N) + \left\langle \ln p(s_{1:T-1}|\mathbf{A},\vec{\pi}) \right\rangle_{q(s_{1:T-1})} \\ &\quad + \ln p(\gamma|N) + \langle \ln p(\vec{x}_{1:T}|s_{1:T-1},\vec{\gamma}) \rangle_{q(s_{1:T-1})} \\ &= -\ln Z + \sum_{j=1}^{N} \Big[\langle \delta_{j,s_1} \rangle_{q(s_{1:T-1})} \ln \pi_j + \ln p(\pi_j) \Big] \\ &\quad + \sum_{j=1}^{N} \sum_{t=1}^{T-2} \left(\ln p(A_{j,:}) + \sum_{k=1}^{N} \left\langle \delta_{j,s_t} \delta_{k,s_{t+1}} \right\rangle_{q(s_{1:T-1})} \ln A_{j,k} \right) \\ &\quad + \sum_{j=1}^{N} \left[\sum_{t=1}^{T-1} \left(\frac{d}{2} \langle \delta_{j,s_t} \rangle_{q(s_{1:T-1})} \ln \frac{\gamma_j}{\pi} - \langle \delta_{j,s_t} \rangle_{q(s_{1:T-1})} (\vec{x}_{t+1} - \vec{x}_t)^2 \gamma_j \right) + \ln p(\gamma_j) \right].\end{aligned} \tag{S18}$$

From this, we can read out the functional form of the variational distribution, which factorize in the same was as the prior. However, we must still choose prior distributions. We adopt a particularly simple form known as conjugate priors[4,6], which enables a simple interpretation of the prior distribution in terms of fictitious 'pseudo'-observations.

Initial state and transition rates: Following earlier work[4,6,7], we choose Dirichlet (a multivariate version of the β distribution) priors for the initial state distribution and for each row of the transition rate matrix[12]. This makes the variational distributions Dirichlet distributions as well (the Dirichlet distribution is its own conjugate). The Dirichlet density function, in this case for $\vec{\pi}$, is

$$q(\vec{\pi}) = \mathrm{Dir}(\vec{\pi}|\vec{w}^{(\vec{\pi})}) = \frac{1}{B(\vec{w}^{(\vec{\pi})})} \prod_j \pi_j^{(w_j^{(\vec{\pi})}-1)}, \tag{S19}$$

with the constraints $0 \leq \pi_j \leq 1$ and $\sum_j \pi_j = 1$, and normalization constant $B(\vec{w}^{(\vec{\pi})}) = \prod_j \Gamma(w_j^{(\vec{\pi})}) / \Gamma(\sum_k w_k^{(\vec{\pi})})$[12]. Inspection of Eq. (S18) reveals that

$$q(\vec{\pi}) = \mathrm{Dir}(\vec{\pi}|\vec{w}^{(\vec{\pi})}), \qquad w_j^{(\vec{\pi})} = \tilde{w}_j^{(\vec{\pi})} + \langle \delta_{j,s_1} \rangle_{q(s)} \tag{S20}$$

$$q(\mathbf{A}) = \prod_j \mathrm{Dir}(A_{j,:}|w_{j,:}^{(\mathbf{A})}), \qquad w_{jk}^{(\mathbf{A})} = \tilde{w}_{jk}^{(\mathbf{A})} + \sum_{t=1}^{T-2} \left\langle \delta_{j,s_t} \delta_{k,s_{t+1}} \right\rangle_{q(s)}, \tag{S21}$$

where $\tilde{w}_j^{(\vec{\pi})}$ and $\tilde{w}_{jk}^{(\mathbf{A})}$ are the Dirichlet parameters of the prior distributions. Note how the prior parameters simply add to the data-dependent terms. This means that the prior distributions can be understood in terms

of 'pseudo-counts', imaginary observations prior to seeing the data in question. The total number of counts (for each distribution) is called the prior strength.

The following averages[6]: will turn out to be useful:

$$\langle \ln \pi_i \rangle_{q(\vec{\pi})} = \psi(w_i^{(\vec{\pi})}) - \psi(w_0^{(\vec{\pi})}), \qquad w_0^{(\vec{\pi})} = \sum_{i=1}^{N} w_i^{(\vec{\pi})}, \tag{S22}$$

$$\langle \ln A_{kj} \rangle_{q(\mathbf{A})} = \psi(w_{kj}^{(\mathbf{A})}) - \psi(w_{k0}^{(\mathbf{A})}), \qquad w_{k0}^{(\mathbf{A})} = \sum_{j=1}^{N} w_{kj}^{(\mathbf{A})}, \tag{S23}$$

$$\pi^*_{i\,q(\vec{\pi})} = \frac{w_i^{(\vec{\pi})} - 1}{w_0^{(\vec{\pi})} - N}, \qquad A^*_{jk\,q(\mathbf{A})} = \frac{w_{jk}^{(\mathbf{A})} - 1}{w_0^{(\mathbf{A})} - N}, \tag{S24}$$

$$\langle \pi_i \rangle_{q(\vec{\pi})} = \frac{w_i^{(\vec{\pi})}}{w_0^{(\vec{\pi})}}, \qquad \langle A_{jk} \rangle_{q(\mathbf{A})} = \frac{w_{jk}^{(\mathbf{A})}}{w_0^{(\mathbf{A})}}, \tag{S25}$$

$$\mathrm{Var}[\pi_i]_{q(\vec{\pi})} = \frac{w_i^{(\vec{\pi})}(w_0^{(\vec{\pi})} - w_i^{(\vec{\pi})})}{(w_0^{(\vec{\pi})})^2(w_0^{(\vec{\pi})} + 1)}, \qquad \mathrm{Var}[A_{jk}]_{q(\mathbf{A})} = \frac{w_{jk}^{(\mathbf{A})}(w_0^{(\mathbf{A})} - w_{jk}^{(\mathbf{A})})}{(w_0^{(\mathbf{A})})^2(w_0^{(\mathbf{A})} + 1)}. \tag{S26}$$

Here, ψ is the digamma function, and $\theta^*_{q(\cdot)}$ denotes the mode (point of maximum density) of the parameter θ in the distribution $q(\cdot)$.

Diffusion constants: The terms involving γ_j in Eq. (S18) are

$$\ln q(\gamma_j) = \text{const.} + \ln p(\gamma_j|N) + \sum_{t=1}^{T-1} \frac{d}{2} \langle \delta_{j,s_t} \rangle \ln \frac{\gamma_j}{\pi} - \langle \delta_{j,s_t} \rangle (\vec{x}_{t+1} - \vec{x}_t)^2 \gamma_j. \tag{S27}$$

Hence, if we choose the prior to be a gamma-distribution, then the variational distribution will be so as well, and we get

$$q(\gamma_j) = \frac{c_j^{n_j}}{\Gamma(n_j)} \gamma_j^{n_j - 1} e^{-c_j \gamma_j}, \tag{S28}$$

with

$$n_j = \tilde{n}_j + \frac{d}{2} \sum_{t=1}^{T-1} \langle \delta_{j,s_t} \rangle, \quad c_j = \tilde{c}_j + \sum_{t=1}^{T-1} \langle \delta_{j,s_t} \rangle (\vec{x}_{t+1} - \vec{x}_t)^2, \tag{S29}$$

and $\tilde{n}_j$, $\tilde{c}_j$ being prior parameters. Again, we will need some averages with respect to $q(\gamma_j)$:

$$\langle \gamma_j \rangle_{q(\gamma_j)} = \frac{n_j}{c_j}, \quad \langle \ln \gamma_j \rangle_{q(\gamma_j)} = \psi(n_j) - \ln c_j, \quad \mathrm{Var}[\gamma_j]_{q(\gamma_j)} = \frac{n_j}{c_j^2}, \tag{S30}$$

and we can also transform back to the diffusion constant $D_j = (4\gamma_j \Delta t)^{-1}$. Using the rules of probability theory, the distribution of D_j is inverse gamma,

$$q(D_j) = \frac{\beta_j^{n_j}}{\Gamma(n_j)} D_j^{-(n_j+1)} e^{-\beta_j/D_j}. \quad \beta_j = c_j/(4\Delta t), \tag{S31}$$

This means that

$$D^*_{j\, q(D_j)} = \frac{c}{4(n_j+1)\Delta t}, \tag{S32}$$

$$\langle D_j \rangle_{q(D_j)} = \frac{c}{4(n_j-1)\Delta t}, \tag{S33}$$

$$\mathrm{std}[D_j]_{q(D_j)} = \frac{\langle D_j \rangle}{\sqrt{n_j - 2}}. \tag{S34}$$

The mean value and standard deviation are only defined if $n_i > 1, 2$ respectively.

The M-step of the iterations, Eq. (S8), thus consists of updating the variational parameter distributions according to equations (S20), (S21), and (S29). These equations in turn contain certain averages of the hidden state distribution $q(s_{1:T-1})$. We now go on and derive the variational distribution for the hidden state and the E-step, which allows us to compute these averages.

4.2 Hidden states

Collecting the terms in Eqs. (S15) and (S13) that depend on the hidden states, we get

$$\begin{aligned} \ln q(s_{1:T-1}) &= -\ln Z_s + \Big\langle \ln p(s_{1:T-1}|\mathbf{A}, \vec{\pi}) \Big\rangle_{q(\mathbf{A})q(\vec{\pi})} + \langle \ln p(\vec{x}_{1:T}|s_{1:T-1}, \vec{\gamma}) \rangle_{q(\vec{\gamma})} \\ &= -\ln Z_s + \sum_{t=1}^{T-1} \ln H_{t,s_t} + \sum_{t=1}^{T-2} \ln Q_{s_t, s_{t+1}}. \end{aligned} \tag{S35}$$

This looks a like the Hamiltonian of a 1-dimensional spin model in statistical physics, with external field $\ln H_{t,j}$ and nearest-neighbor coupling $\ln Q_{jk}$ given by

$$\ln H_{t,j} = \delta_{1,t} \underbrace{\big[\psi(w_j^{(\vec{\pi})}) - \psi(w_0^{(\vec{\pi})})\big]}_{\text{initial state distribution}} + \frac{d}{2}\big(\psi(n_j) - \ln \pi c_j\big) - \frac{n_j}{c_j}(\vec{x}_{t+1} - \vec{x}_t)^2, \tag{S36}$$

$$\ln Q_{j,k} = \psi(w_{jk}^{(\mathbf{A})}) - \psi(w_{j0}^{(\mathbf{A})}). \tag{S37}$$

For this distribution, we also need a couple of expectation values to feed back in the next iteration of the parameter distributions, namely

$$\langle \delta_{j,s_t} \rangle_{q(s)} = p(s_t = j), \quad \big\langle \delta_{j,s_t} \delta_{k,s_{t+1}} \big\rangle_{q(s)} = p(s_{t+1} = k | s_t = j). \tag{S38}$$

These can be computed by a dynamic programming trick[3,4,6], known in the HMM context as the forward-backward or Baum-Welch algorithm[13,14]. As a side effect, one also obtains the normalization constant Z_s, which will be needed below when computing the lower bound F.

Multiple trajectories: To analyze many trajectories, one tries to optimize the sum of the lower bounds for each trajectory with a single model. For the EM-iterations, this means that the parameters in the variational distributions get contributions from each trajectory that are just summed up, *i.e.* averages over the hidden state distribution gets extended with a summation over M trajectories of length T_m as well, *e.g.* in Eq. (S21),

$$w_{jk}^{(\mathbf{A})} = \tilde{w}_{jk}^{(\mathbf{A})} + \sum_{m=1}^{M} \sum_{t=1}^{T_m-2} \left\langle \delta_{j,s_t^m} \delta_{k,s_{t+1}^m} \right\rangle_{q(s)}. \tag{S39}$$

We get a variational distribution over the hidden states in each trajectory, where the coupling constants Q_{jk} and external fields H_{tj} are given by the same parameter distributions (but differ in the contributions from data).

4.3 Lower bound

The lower bound F can be computed just after the E-step. Substituting our variational ansatz $q(\theta, s) = q(\theta)q(s)$ into our general expression for F, Eq. (S7), we can exploit the fact that $q(s)$ and $q(\theta)$ are normalized probability distributions, and rewrite it in the form

$$\begin{aligned} F[q(\theta)q(s), x] &= \int d\theta \sum_s \Big[q(\theta)q(s) \ln p(x, s|\theta, N)p(\theta|N) - q(\theta)q(s) \ln q(\theta)q(s) \Big] \\ &= \sum_s q(s) \Big[\langle \ln p(x, s|\theta, N) \rangle_{q(\theta)} - \ln q(s) \Big] + \int d\theta q(\theta) \Big[\ln p(\theta|N) - \ln q(\theta) \Big]. \end{aligned} \tag{S40}$$

Just after the E-step however, $q(s)$ is given by Eq. (S9), which substituted in the above expressions leads to cancellations, leaving us with

$$F[q(\theta)q(s), x] = \ln Z_s - \int d\theta q(\theta) \ln \frac{q(\theta)}{p(\theta|N)}. \tag{S41}$$

The first term is just the normalization constant in Eq. (S9), which comes out as a by-product of the forward-backward algorithm. The second term is the negative Kullback-Leibler divergence of the variational parameter distribution with respect to the prior. In our case, this distribution factorizes

as in Eq. (S17), and we get a sum of separate and tractable contributions,

$$\int d\theta q(\theta) \ln \frac{q(\theta)}{p(\theta|N)} = \int d\vec{\pi} q(\vec{\pi}) \ln \frac{q(\vec{\pi})}{p(\vec{\pi}|N)} + \sum_j \int dA_{j,:} q(A_{j,:}) \ln \frac{q(A_{j,:})}{p(A_{j,:}|N)} + \sum_j \int d\gamma_j q(\gamma_j) \ln \frac{q(\gamma_j)}{p(\gamma_j|N)}. \quad \text{(S42)}$$

For the initial state distribution, we get

$$\int d\vec{\pi} q(\vec{\pi}) \ln \frac{q(\vec{\pi})}{p(\vec{\pi}|N)} = \left\langle \ln \left(\frac{\Gamma(w_0^{(\vec{\pi})})}{\Gamma(\tilde{w}_0^{(\vec{\pi})})} \prod_{j=1}^{N} \frac{\Gamma(\tilde{w}_j^{(\vec{\pi})})}{\Gamma(w_j^{(\vec{\pi})})} \pi_j^{w_j^{(\vec{\pi})} - \tilde{w}_j^{(\vec{\pi})}} \right) \right\rangle_{q(\vec{\pi})}$$
$$= \ln \frac{\Gamma(w_0^{(\vec{\pi})})}{\Gamma(\tilde{w}_0^{(\vec{\pi})})} + \sum_{j=1}^{N} \left[\ln \frac{\Gamma(\tilde{w}_j^{(\vec{\pi})})}{\Gamma(w_j^{(\vec{\pi})})} + (w_j^{(\vec{\pi})} - \tilde{w}_j^{(\vec{\pi})}) \big(\psi(w_j^{(\vec{\pi})}) - \psi(w_0^{(\vec{\pi})}) \big) \right], \quad \text{(S43)}$$

where we used Eq. (S22) for $\langle \ln \pi_i \rangle_{q(\vec{\pi})}$. The terms from the transition matrix rows are also Dirichlet distributed, and come out as

$$\int dA_{j,:} q(A_{j,:}) \ln \frac{q(A_{j,:})}{p(A_{j,:}|N)} = \left\langle \ln \left(\frac{\Gamma(w_{j0}^{(\mathbf{A})})}{\Gamma(\tilde{w}_{j0}^{(\mathbf{A})})} \prod_{k=1}^{N} \frac{\Gamma(\tilde{w}_{jk}^{(\mathbf{A})})}{\Gamma(w_{jk}^{(\mathbf{A})})} A_{jk}^{w_{jk}^{(\mathbf{A})} - \tilde{w}_{jk}^{(\mathbf{A})}} \right) \right\rangle_{q(\mathbf{A})}$$
$$= \ln \frac{\Gamma(w_{j0}^{(\mathbf{A})})}{\Gamma(\tilde{w}_{j0}^{(\mathbf{A})})} + \sum_{k=1}^{N} \left[\ln \frac{\Gamma(\tilde{w}_{jk}^{(\mathbf{A})})}{\Gamma(w_{jk}^{(\mathbf{A})})} + (w_{jk}^{(\mathbf{A})} - \tilde{w}_{jk}^{(\mathbf{A})}) \big(\psi(w_{jk}^{(\mathbf{A})}) - \psi(w_{j0}^{(\mathbf{A})}) \big) \right]. \quad \text{(S44)}$$

Finally, the terms for the precision parameter γ_j can be written in the form

$$\int d\gamma_j q(\gamma_j) \ln \frac{q(\gamma_j)}{p(\gamma_j|N)} = \left\langle \ln \frac{c_j^{n_j} \gamma_j^{n_j - 1} e^{-c_j \gamma_j} \Gamma(\tilde{n}_j)}{\tilde{c}_j^{\tilde{n}_j} \gamma_j^{\tilde{n}_j - 1} e^{-\tilde{c}_j \gamma_j} \Gamma(n_j)} \right\rangle_{q(\gamma_j)}$$
$$= n_j \ln \frac{c_j}{\tilde{c}_j} - \ln \frac{\Gamma(n_j)}{\Gamma(\tilde{n}_j)} + (n_j - \tilde{n}_j) \psi(n_j) - n_j \Big(1 - \frac{\tilde{c}_j}{c_j} \Big), \quad \text{(S45)}$$

where in the last step, we substituted Eq. (S30) for $\langle \gamma_j \rangle_{q(\gamma_j)}$ and $\langle \ln \gamma_j \rangle_{q(\gamma_j)}$.

4.4 Matlab notation

For future reference, we end by listing a translation table between the notation used in this derivation and the variable names used in the VB3 code, with the matlab model object named `W`.

- The parameters of the parameter variational distributions are collected in the `W.M` and `W.PM` fields. These are the only fields needed to start iterating, all the rest are computed by the algorithm. If the `W.E` field is present, then the `W.M` field is overwritten in the first iteration (and hence the `W.E` field must be deleted if one wants to use the `W.M` field to parametrize an initial guess).

Matlab VB3	**This note**	**Matlab VB3**	**This note**	**Eq.**
`W.M.wPi(i)`	$w_i^{(\vec{\pi})}$	`W.PM.wPi(i)`	$\tilde{w}_i^{(\vec{\pi})}$	(S20)
`W.M.wA(j,k)`	$w_{jk}^{(\mathbf{A})}$	`W.PM.wA(j,k)`	$\tilde{w}_{jk}^{(\mathbf{A})}$	(S21)
`W.M.n(j)`	n_j	`W.PM.n(j)`	$\tilde{n}_j$	(S29)
`W.M.c(j)`	c_j	`W.PM.c(j)`	$\tilde{c}_j$	

- Expectation values computed in the E-step are in the `W.E` fields. Each trajectory gets its own field, and for trajectory `m`, the notation is

Matlab VB3	**This note**	**Eq.**
`W.E(m).wPi(j)`	$\langle \delta_{j,s_1} \rangle$	(S20)
`W.E(m).wA(j,k)`	$\sum_{t=1}^{T-2} \langle \delta_{j,s_t} \delta_{k,s_{t+1}} \rangle$	(S21)
`W.E(m).n(j)`	$\frac{d}{2} \sum_{t=1}^{T-1} \langle \delta_{j,s_t} \rangle$	(S29)
`W.E(m).c(j)`	$\sum_{t=1}^{T-1} \langle \delta_{j,s_t} \rangle (\vec{x}_{t+1} - \vec{x}_t)^2$	(S29)

- Misc fields:

`W.N`	Number of hidden states N.
`W.F`	Total lower bound F.
`W.Fterms`	Various contributions to F (for debugging).
`W.T(j)`	Length of trajectory j. Note that this counts the number of steps, and hence trajectory j will contain `W.T(j)`+1 particle positions.

- Interesting estimates are supplied to describe the output and make sense of the converged model and the data. They divided into two fields, `W.est` for quantities that are small and cheap to compute (and hence are given every time), while `W.est2` contain quantities that are either large or expensive to compute, and therefore only supplied when the iterator algorithm is called with the `'estimate'` argument. When given, the index `m` refers to trajectory number.

Matlab VB3	**This note**	**Eq.**
`W.est.lnQ(j,k)`	$\ln Q_{j,k}$	(S35)
`W.est.Q(j,k)`	$Q_{j,k} / \max_{j,k}(Q_{j,k})$	
`W.est.Ts(j)`	$\sum_t \langle \delta_{j,s_t} \rangle$	
`W.est.Ps`	`W.est.Ts/sum(W.est.Ts)`	
`W.est.Amean(j,k)`	$\langle A_{jk} \rangle_{q((A)}$	(S25)
`W.est.Amode(j,k)`	$A^*_{jk\ q(\mathbf{A})}$	(S24)
`W.est.Astd(j,k)`	$\mathrm{std}[A_{jk\ q(\mathbf{A})}]$	(S26)
`W.est.lnAmean(k,j)`	$e^{\texttt{W.est.lnAmean}} = \langle \mathbf{A} \rangle$	approx. rate matrix $[\Delta t^{-1}]$
`W.est.lnAmode(k,j)`	$e^{\texttt{W.est.lnAmode}} = \mathbf{A}^*$	approx. rate matrix $[\Delta t^{-1}]$
`W.est.dwellMean(j)`	$(1 - \langle A_{jj} \rangle)^{-1}$	dwell time
`W.est.dwellMode(j)`	$(1 - A^*_{jj})^{-1}$	dwell time
`W.est.gMean(j)`	$\langle \gamma_j \rangle_{q(\vec{\gamma})}$	(S30)

Matlab VB3	**This note**	**Eq.**
`W.est2.lnH{m}(t,j)`	$\ln H_{t,j}$	(S35)
`W.est2.H{m}(t,j)`	$H_{t,j} / \max_k(H_{t,k})$	
`W.est2.viterbi{m}`		Viterbi path, trj. m.
`W.est2.sMaxP{m}(j)`	$\mathrm{argmax}_{s_t^m} \left\langle \delta_{j,s_t^m} \right\rangle$	Most likely states.
`W.est.pst{m}(t,j)`	$\left\langle \delta_{j,s_t^m} \right\rangle$	Occupation probability $p(s_t^m = j)$.

References

[1] Fredrik Persson, Martin Lindn, Cecilia Unoson, and Johan Elf. Extracting intracellular diffusive states and transition rates from single-molecule tracking data. *Nat. Meth.*, 10(3):265–269, 2013. doi: 10.1038/nmeth.2367.

[2] D. MacKay. *Information theory, inference, and learning algorithms*. Cambridge University Press, Cambridge UK, 2003.

[3] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, New York USA, 2006.

[4] D. J. C. MacKay. Ensemble learning for hidden Markov models. Accessed Feb 15 2011, 1997. URL `http://www.inference.phy.cam.ac.uk/mackay/abstracts/ensemblePaper.html`.

[5] Z. Ghahramani. An introduction to hidden Markov models and Bayesian networks. In *Hidden Markov models: applications in computer vision*, pages 9 – 42. World Scientific Publishing Co., Hackensack USA, 2002. URL `http://learning.eng.cam.ac.uk/zoubin/papers/`.

[6] Matthew Beal. *Variational Algorithms for approximate Bayesian inference*. PhD thesis, University of Cambridge, UK, 2003.

[7] J. E. Bronson, J. Y. Fei, J. M. Hofman, R. L. Gonzalez, and C. H. Wiggins. Learning rates and states from biophysical time series: A Bayesian approach to model selection and single-molecule FRET data. *Biophys. J.*, 97(12):3196 – 3205, 2009.

[8] J. E. Bronson, J. M. Hofman, J. Fei, R. L. Gonzalez, and C. H. Wiggins. Graphical models for inferring single molecule dynamics. *BMC Bioinformatics*, 11 Suppl 8:S2, 2010.

[9] Sean R Eddy. What is Bayesian statistics? *Nat. Biotech.*, 22(9):1177–1178, 2004.

[10] P. J. Green. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biomet.*, 82(4):711 – 732, 1995.

[11] C. P. Robert, T. Rydén, and D. M. Titterington. Bayesian inference in hidden Markov models through the reversible jump Markov chain Monte Carlo method. *J. Roy. Stat. Soc. B*, 62(1):57 – 75, 2000.

[12] Wikipedia. Dirichlet distribution — wikipedia, the free encyclopedia, 2012. URL `\url{http://en.wikipedia.org/w/index.php?title=Dirichlet_distribution&oldid=478066050}`. [Online; accessed 26-February-2012].

[13] L.R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *P. IEEE*, 77(2):257 – 286, 1989. doi: 10.1109/5.18626.

[14] L.E. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. *Inequalities*, 3:1 – 8, 1972.