State arrays for single particle tracking (saSPT) - User Guide

Alec Heckert, Liza Dahal, Xavier Darzacq & Robert Tjian

May 2021

saSPT is a tool to extract information from trajectories collected in live cell stroboscopic photoactivated single particle tracking (spaSPT) experiments. The output of saSPT is a distribution over one or more parameters that govern the particles' motion, enabling the user to identify diffusive states and their properties in complex and noisy data. This user guide and the rest of the code can be found at `https://gitlab.com/alecheckert/saspt`.

If using saSPT for academic research, please cite the saSPT research article:

Alec Heckert, Liza Dahal, Robert Tjian, Xavier Darzacq. Recovering mixtures of fast diffusing states from short single particle trajectories (preprint). *bioRxiv* (2021). `https://doi.org/10.1101/2021.05.03.442482`

While this User Guide contains a brief section on the algorithms behind state arrays, the research article cited above contains substantially more detail and should be consulted for derivations of the algorithms outlined here and for a treatment of the defocalization problem in SPT, which we largely gloss over here.

Contents

1 saSPT user manual

1.1 Quick start

See `examples/example_state_arrays.py` in this repo for an example of how to run saSPT on trajectories.

1.2 What does saSPT do?

saSPT aims to provide a simple and intuitive tool to learn about the mobility of particles in a tracking experiment while accounting for some known biases in SPT. saSPT's underlying model assumes that trajectories come from a mixture of underlying *diffusive states*. Each state can be considered a possible model for the particle's motion, and has an associated set of one or more parameters. An example parameter is the diffusion coefficient for regular Brownian motion (RBM), which parametrizes the variance of the jumps made by a particle undergoing RBM.

This kind of mixture model is very popular, and many methods attempt to infer some or all of the following:

1. the number of states
2. the parameters for each state
3. the occupation of each state (its relative abundance)
4. the origin state for each observed trajectory

saSPT relies on a kind of inference that favors sparse models with a small number of states rather than more complex alternatives. As a result, instead of attempting to get the number of states or the state parameters directly, it uses a large "grid" of state parameters, the occupations of most of which are driven to zero in the course of the algorithm. As a result, we only infer (3) and (4) in the list above. The output of inference is a *posterior distribution* over the state occupations and the trajectory-state assignments, which captures the information inferred about the state occupations and trajectory-state assignments in addition to providing some idea of the uncertainty on these parameters.

saSPT takes a set of trajectories as a CSV and returns the posterior distribution as a CSV, `numpy.ndarray`, or `pandas.DataFrame`. It can also generate some plots that summarize the outcome of the inference, and performs a limited number of related analyses that are useful to judge experiment-to-experiment variability, spatial and temporal variation in the diffusive profile of an SPT target, and other effects of interest for inferring diffusive states.

1.3 What doesn't saSPT do?

- saSPT is not a tool for detection or tracking. It expects you to detect and track fluorescent emitters by another tracking algorithm.
- Apart from a rudimentary filter on the starting frame index, saSPT does not provide any options to filter trajectories prior to inferring the posterior.
- saSPT has no way to detect tracking errors or misdetections in SPT data. It does not check the quality of your input data at all, and will analyze exactly what you give it, regardless of whether the trajectories are high-confidence or junk. The output of saSPT is only as good as the quality of the input trajectories.
- saSPT requires that you provide the parameters for your tracking experiment, including the frame interval, microscope focal depth, and pixel size. Some likelihood functions (for instance, `gamma` or `fbme` assume that you have measured the approximate localization error in advance.
- saSPT does not provide a graphical user interface (GUI). The usage is intended to be simple enough for a basic knowledge of Python to suffice

when writing scripts that use it. See the scripts in `examples` for examples of how to use saSPT, and section 1.7 for more details.

1.4 Installation

saSPT is a Python package that requires `numpy`, `pandas`, `tqdm`, `scipy`, `matplotlib`, and optionally `matplotlib_scalebar` (if you want scalebars on your plots). If you are using `conda` as a package manager, an environment with most of these dependencies can be found here. The exception is `matplotlib_scalebar`, which can get via `pip` (`pip install matplotlib_scalebar`).

Once you have the required dependencies, you can install saSPT with these steps:

1. Clone the saSPT repo:

```
git clone https://gitlab.com/alecheckert/saspt; cd saspt
```

2. Install saSPT:

```
python setup.py develop
```

3. Run some examples to make sure it works:

```
python examples/example_state_arrays.py
```

1.5 Expected input format

saSPT expects trajectories as a CSV file, each row encoding a separate detection, with the following columns at minimum:

- `y`: the y-coordinate of the detection in pixels
- `x`: the x-coordinate of the detection in pixels
- `frame`: the frame index of the detection in pixels (an integer)
- `trajectory`: the index of the trajectory to which this detection has been assigned (an integer)

Variations on these names are *not* tolerated. If other columns are present, they are ignored by saSPT. The order of the columns doesn't matter.

Examples of the expected input for saSPT can be found in the `examples/u2os_ht_nls_7.48ms` and `examples/u2os_rara_ht_7.48ms` directories.

1.6 Examples

A set of example scripts using saSPT can be found in the `examples` folder of the repo. These use two experimental saSPT datasets collected by the authors. The expected output for each script is stored under the `examples/expected_outputs` directory.

1.7 The StateArray object

This section describes the `saspt.StateArray` object in more detail. This class is responsible for inferring the posterior of a state array given an observed set of trajectories, and also provies other useful methods and attributes.

1.7.1 Instantiating a StateArray

To instantiate a StateArray, import the class and pass it your trajectories along with the parameters for your tracking experiment:

```
import numpy as np
from saspt import load_tracks, StateArray

# Load some trajectories
trajectories = load_tracks('path/to/some/trajectory/CSVs')

# Make a StateArray
sa = StateArray(
    trajectories,
    likelihood="rbme",   # the likelihood function to use
    pixel_size_um=0.16,  # size of pixels in microns
    frame_interval=0.00748, # frame interval in seconds
    start_frame=1000,    # disregard trajectories before frame 1000
    dz=0.7,              # focal depth is ~0.7 microns
    max_iter=500         # do 500 iterations of inference
)
```

The ‘likelihood‘ argument defines the kind of state grid over which we infer the posterior. saSPT supports four kinds of likelihood functions: `rbme`, `gamma`, `rbme_marginal`, and `fbme`. All of the experiments in the saSPT paper used the ‘rbme‘ likelihood, and the other options may be considered experimental.

The four supported likelihood functions correspond to the following parameters:

- `rbme`: the diffusion coefficient for regular Brownian motion and the localization error
- `gamma`: the diffusion coefficient for regular Brownian motion (gamma approximation)
- `rbme_marginal`: the diffusion coefficient for regular Brownian motion, marginalized on localization error
- `fbme`: the diffusion coefficient and Hurst parameter for fractional Brownian motion with known localization error

When getting started with saSPT, we recommend sticking to the `rbme` likelihood as it is by far the best tested and can be compared with previous results.

1.7.2 Attributes

The `StateArray` class has the following attributes:

- `support`: a tuple of `numpy.ndarray` that give the set of parameters on which this state array is defined. For example, if we choose `likelihood = 'rbme'`, then the first element of this tuple is a set of diffusion coefficients while the second is a set of localization errors.
- `track_indices`: a `numpy.ndarray` that gives the trajectory index for each (non-singleton) trajectory considered by the state array. These indices are obtained directly from the input `trajectories` dataframe. Note that one trajectory index can potentially be repeated multiple times in `track_indices` if that trajectory was split into smaller pieces prior to inference (see “splitsize” in 1.7.4).
- `n_jumps`: a `numpy.ndarray` that gives the number of jumps observed in each (non-singleton) trajectory. This also defines the weight that the trajectory contributes to inference of state occupations.
- `L`: a `numpy.ndarray`, the raw likelihood function evaluated on each trajectory for each point in the state array. For example, if `likelihood = 'rbme'`,

then `L[i,j,k]` gives the likelihood function evaluated on trajectory `i` at diffusion coefficient `j` and localization error `k`.

- `aggregated_likelihood`: a `numpy.ndarray` that gives the likelihood function marginalized over all trajectories, then normalized. This provides a naïve estimate of the state occupations, and can be considered a drop-in replacement for MSD histograms that is not prone to the same sampling biases.
- `posterior`: the posterior distribution as a tuple of `numpy.ndarray`. The first element is the posterior distribution over the trajectory-state assignments (expressed as a categorical probability for each trajectory), the second element is the posterior distribution over the state occupations (expressed as the parameter to a Dirichlet distribution), and the third element is the mean of the posterior over state occupations.

The most important output of the `StateArray` is the third element of `StateArray.posterior`, the posterior mean of the state occupations. Each point in this array gives the estimated occupation of the state with the corresponding parameters.

For example, say we wanted to marginalize the posterior distribution on localization error (treating it as a nuisance parameter) and then get the estimated fraction of trajectories with diffusion coefficients between 0.1 and 1.0 μm^2 s^{-1}. Then we could do

```
# Get the support for this state array
diff_coefs, loc_errors = sa.support

# Estimate state occupations by taking the posterior mean
_, _, posterior_mean = sa.posterior

# Marginalize out localization error
posterior_mean_marg = posterior_mean.sum(axis=1)

# Calculate the total state occupation between diffusion
# coefficients 0.1 and 1.0 squared microns per second
print(posterior_mean_marg[np.logical_and(
    diff_coefs >= 0.1,
    diff_coefs <= 1.0
)])
```

1.7.3 Defining the support for a state array directly

The support for the state array can be set manually during `StateArray` instantiation:

```
# Define the set of diffusion coefficients on which to define
# the state array
diff_coefs = np.logspace(-2.0, 2.0, 301)

# Define the set of localization errors on which to define the
# state array
loc_errors = np.linspace(0.01, 0.62, 0.02)

# Instantiate the StateArray on this support
sa = StateArray(
    trajectories,
    likelihood="rbme",
    pixel_size_um=0.16,
    frame_interval=0.00748,
    start_frame=1000,
    dz=0.7,
    max_iter=500,
    diff_coefs=diff_coefs,
    loc_errors=loc_errors
)
```

1.7.4 Other instantiation parameters

The `StateArray` class also provides the user the ability to set a variety of other parameters that affect inference. Unless there is good reason or the user is experienced, it is recommend *not* to change these parameters.

These additional parameters are:

- `max_jumps_per_track`: the maximum number of jumps to consider from any given trajectory. If a trajectory surpasses this number, jumps after the limit are ignored (the trajectory is truncated). By default, this is set to `None` and has no effect.
- `splitsize`: the size of the pieces into which trajectories are split prior to inference (in # of jumps). Since the inferential weight of each trajectory scales with the number of jumps, trajectory splitting prevents any single trajectory from exerting too strong an influence on posterior estimation,

which is especially important in the presence of tracking errors. It also improves accuracy in the presence of state transitions that have kinetics similar to the frame interval. The default `splitsize` is 8.

- `pseudocount_frac`: the number of pseudocounts per state in the prior, expressed as a fraction of the total number of jumps in the set of trajectories. The larger the value of `pseudocount_frac`, the harder incoming trajectories have to work in order to cause the posterior distribution to depart from the prior. In addition, the number of pseudocounts per state is never allowed to dip below a hard value set by `saspt.constants.MIN_PSEUDOCOUNTS`. This is a safety feature to prevent spurious results when saSPT is run with too few trajectories.
- `convergence`: criterion for convergence in terms of the parameter to the updating Dirichlet distribution over state occupations. By default 0, so that we run the maximum number of iterations (`max_iter`).

1.8 Units

Where unspecified, units in saSPT are always the following:

- *time*: seconds
- *distance*: μm ("microns")
- *diffusion coefficient*: $\mu\text{m}^2\ \text{s}^{-1}$
- *localization error*: μm (root 1D localization variance)

2 Description of algorithm

In this section, we describe the underlying model for saSPT and outline the inference method.

2.1 State array model

saSPT represents a set of trajectories as a *mixture model*. In this kind of model, each trajectory is a random variable that is generated in a two-step process:

1. First, draw a random *state* from a distribution over states. Each state has an associated set of *state parameters* - for example, the diffusion coefficient for a regular Brownian motion (RBM) state. We represent the fractional occupation of state j as τ_j and its state parameters as θ_j.
2. Next, generate a random trajectory given those state parameters. For instance, in the case of RBM, generate a trajectory with the diffusion coefficient corresponding to the state selected in step (1).

We can represent the trajectory-state assignment in step (1) as a random matrix $\mathbf{Z}$, where $Z_{ij} = 1$ if trajectory i comes from state j and $Z_{ij} = 0$ otherwise. Each row (specific to a trajectory i) has exactly one value equal to 1.

Without knowledge of $\mathbf{Z}$, the probability to draw some trajectory x given state occupations $\boldsymbol{\tau}$ and state parameters $\boldsymbol{\theta}$ is proportional to

$$p\left(x|\boldsymbol{\tau},\boldsymbol{\theta}\right) \propto \sum_{j=1}^{K} \tau_j f_X(x|\theta_j) \tag{1}$$

where K is the total number of states. $f_X(x|\theta_j)$ is the *trajectory likelihood*, providing the probability density over trajectories given state parameters θ_j, and is determined by the diffusion model (Brownian, fractional Brownian, etc.).

In a real SPT experiment, we observe a large number of trajectories $\mathbf{X} = (x_1, ..., x_N)$, where each x_i is a separate trajectory (using whatever representation is convenient for the likelihood function). Our goal is to learn about K, $\boldsymbol{\tau}$, $\boldsymbol{\theta}$, and $\mathbf{Z}$ given this set of trajectories. While there are many ways to do this, the approach taken by Bayesian statistics is to let $\boldsymbol{\tau}$, $\boldsymbol{\theta}$, and $\mathbf{Z}$ be random variables and then calculate their conditional distribution given $\mathbf{X}$ using Bayes' theorem:

$$\begin{aligned} p\left(\boldsymbol{\tau},\boldsymbol{\theta},\mathbf{Z}|\mathbf{X}\right) &= \frac{p\left(\mathbf{X}|\boldsymbol{\tau},\boldsymbol{\theta},\mathbf{Z}\right) p\left(\boldsymbol{\tau},\boldsymbol{\theta},\mathbf{Z}\right)}{p\left(\mathbf{X}\right)} \\ &= \frac{p\left(\mathbf{X}|\boldsymbol{\tau},\boldsymbol{\theta},\mathbf{Z}\right) p\left(\boldsymbol{\tau},\boldsymbol{\theta},\mathbf{Z}\right)}{\int p\left(\mathbf{X}|\boldsymbol{\tau},\boldsymbol{\theta},\mathbf{Z}\right) p\left(\boldsymbol{\tau},\boldsymbol{\theta},\mathbf{Z}\right) \, \mathrm{d}\boldsymbol{\tau} \, \mathrm{d}\boldsymbol{\theta} \, \mathrm{d}\mathbf{Z}} \end{aligned} \tag{2}$$

(As it stands, this equation assumes knowledge of K, the number of states. We could incorporate K directly into this equation as another parameter, or use posterior criteria like the evidence lower bound (ELBO) to determine K from several runs of inference. As we will see, state arrays assume a large constant K and allow states to be "weeded out" by the inference routine.)

Once we have this conditional distribution, we can then derive a working estimate for $\boldsymbol{\theta}$, $\boldsymbol{\tau}$, and $\mathbf{Z}$ by taking the mean of this conditional distribution:

$$\boldsymbol{\tau}_{\text{est}}, \boldsymbol{\theta}_{\text{est}} = \mathbb{E}\left[\boldsymbol{\tau}, \boldsymbol{\theta} | \mathbf{X}\right]$$

There are other ways to derive a working estimate, but we use the conditional mean because it is fairly conservative.

2.2 Trajectory likelihoods in saSPT

The choice of trajectory likelihood function in equation 1 reflects what we want to learn about the particles in the experiment. For instance, if we wanted to determine whether the motion of a particle is directional or adirectional, we might choose a likelihood function with a parameter that determines the directionality of the motion.

A very common trajectory likelihood function for pure diffusive mixtures (without any net direction to the motion) is *regular Brownian motion* (RBM), first derived in a physical context by Einstein in 1905. RBM has a single parameter, the diffusion coefficient, that parametrizes the variance of the jumps made by a diffusing particle over subsequent frame intervals.

Specifically, if $\mathbf{X}$ and $\mathbf{Y}$ are the x- and y-coordinates of a trajectory generated by RBM with diffusion coefficient D, where $X_0 = Y_0 = 0$, that is observed at regular frame intervals of Δt, then

$$\begin{aligned} \mathbf{X}, \mathbf{Y} &\sim \mathcal{N}\left(\mathbf{0}, \boldsymbol{\Gamma}^{(\text{pos})}\right) \\ \Gamma_{ij}^{(\text{pos})} &= 2D\Delta t \min\left(i, j\right) \\ \text{Cov}\left(\mathbf{X}, \mathbf{Y}\right) &= \mathbf{0} \end{aligned} \tag{3}$$

where $\mathcal{N}\left(\mathbf{0}, \boldsymbol{\Gamma}^{(\text{pos})}\right)$ is a multivariate normal random variable with mean zero (no net drift) and covariance matrix $\boldsymbol{\Gamma}^{(\text{pos})}$.

Equation 3 represents the trajectory as a set of *coordinates* in x and y. It is also possible to represent the trajectory as a set of *jumps* along the x and y axis. A trajectory with n coordinates along one axis will have $n - 1$ jumps along that

axis. If we let $\Delta\mathbf{X}$ and $\Delta\mathbf{Y}$ be the jumps along the x and y axes respectively, then

$$\begin{aligned}\Delta\mathbf{X}, \Delta\mathbf{Y} &\sim \mathcal{N}\left(\mathbf{0}, \mathbf{\Gamma}^{(\text{jumps})}\right) \\ \Gamma_{ij}^{(\text{jumps})} &= \begin{cases} 2D\Delta t & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}\end{aligned} \tag{4}$$

Because this covariance matrix is diagonal, this likelihood can be represented in a simpler way. Let S_i be the sum of squared jumps for trajectory i, so that if there are L_i jumps in trajectory i, then

$$S_i = \sum_{n=1}^{L_i} \left(\Delta X_n^2 + \Delta Y_n^2\right)$$

Then

$$S_i \sim \text{Gamma}\left(L_i, 4D\Delta t\right)$$

where $\text{Gamma}\left(L_i, 2D\Delta t\right)$ is a gamma distribution, corresponding to the PDF

$$f_{S_i}(s|D) = \begin{cases} \frac{s^{L_i-1}e^{-\frac{s}{4D\Delta t}}}{\Gamma(L_i)(4D\Delta t)^{L_i}} & \text{if } s \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

We rarely measure pure RBM in SPT experiments due to *localization error*, the uncertainty associated with the measurement of a particle's position. If we model localization error as a normally distributed random variable $\mathbf{W}$ with mean 0 and variance σ_{loc}^2, then an RBM with localization error (RBME) can be defined as $\tilde{\mathbf{X}} = \mathbf{X} + \mathbf{W}$, where

$$\mathbf{W} \sim \mathcal{N}\left(\mathbf{0}, \sigma_{\text{loc}}^2\mathbf{I}\right)$$

Then

$$\begin{aligned}\tilde{\mathbf{X}}, \tilde{\mathbf{Y}} &\sim \mathcal{N}\left(\mathbf{0}, \tilde{\mathbf{\Gamma}}^{(\text{pos})}\right) \\ \tilde{\Gamma}_{ij}^{(\text{pos})} &= \begin{cases} 2D\Delta t \min(i,j) + \sigma_{\text{loc}}^2 & \text{if } i = j \\ 2D\Delta t \min(i,j) & \text{otherwise} \end{cases}\end{aligned} \tag{6}$$

and the corresponding jump likelihoods are

$$\begin{aligned}\Delta\tilde{\mathbf{X}}, \Delta\tilde{\mathbf{Y}} &\sim \mathcal{N}\left(\mathbf{0}, \tilde{\mathbf{\Gamma}}^{(\text{jump})}\right) \\ \tilde{\Gamma}_{ij}^{(\text{jump})} &= \begin{cases} 2(D\Delta t + \sigma_{\text{loc}}^2) & \text{if } i = j \\ -\sigma_{\text{loc}}^2 & \text{if } |i-j| = 1 \\ 0 & \text{otherwise} \end{cases}\end{aligned} \tag{7}$$

so that the full PDF for a trajectory with jumps $\Delta\tilde{\mathbf{X}}$, $\Delta\tilde{\mathbf{Y}}$ is

$$f_{\Delta\tilde{\mathbf{X}},\Delta\tilde{\mathbf{Y}}}\left(\mathbf{x},\mathbf{y}|D,\sigma^2_{\text{loc}}\right) = \frac{\exp\left(-\frac{1}{2}\left(\mathbf{x}^T\tilde{\mathbf{\Gamma}}^{-1}\mathbf{x} + \mathbf{y}^T\tilde{\mathbf{\Gamma}}^{-1}\mathbf{y}\right)\right)}{2\pi\det\left(\tilde{\mathbf{\Gamma}}^{(\text{jump})}\right)} \tag{8}$$

This corresponds to the `rbme` likelihood in saSPT.

Notice that, unlike in the case of RBM without error, the off-diagonal components of the covariance matrix are nonzero. If $\sigma^2_{\text{loc}} << D\Delta t$, then we can neglect these off-diagonal terms to yield an approximate likelihood analogous to equation 5:

$$f_{S_i}(s|D,\sigma^2_{\text{loc}}) = \begin{cases} \frac{s^{L_i-1}e^{-\frac{s}{4(D\Delta t+\sigma^2_{\text{loc}})}}}{\Gamma(L_i)(4D\Delta t+\sigma^2_{\text{loc}})^{L_i}} & \text{if } s \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

This corresponds to the `gamma` likelihood in saSPT.

Finally, we can venture outside the realm of RBM by considering other types of motion. One of the most useful extensions of RBM is *fractional Brownian motion* (FBM), which allows for nonzero covariance between the jumps (outside of simple localization error). Specifically, FBM endows the motion with an additional parameter called the *Hurst parameter*, and the jump likelihood

$$\begin{aligned} \Delta\tilde{\mathbf{X}},\Delta\tilde{\mathbf{Y}} &\sim \mathcal{N}\left(\mathbf{0},\tilde{\mathbf{\Gamma}}^{(\text{fbm jumps})}\right) \\ \tilde{\Gamma}^{(\text{fbm jumps})}_{ij} &= \begin{cases} g(i,j)+2\sigma^2_{\text{loc}} & \text{if } i=j \\ g(i,j)-\sigma^2_{\text{loc}} & \text{if } |i-j|=1 \\ g(i,j) & \text{otherwise} \end{cases} \\ g(i,j) &= D\Delta t^{2H}\left(|i-j+1|^{2H}+|i-j-1|^{2H}-2\,|i-j|^{2H}\right) \end{aligned} \tag{10}$$

This corresponds to the `fbme` likelihood in saSPT.

While the underlying representation for a trajectory is usually the set of jumps $\Delta\tilde{\mathbf{X}}$ and $\Delta\tilde{\mathbf{Y}}$, for the purposes of simplicity in this manual we often write the i^{th} trajectory simply as X_i.

2.3 Inferring the posterior distribution

The problem with equation 2 is that the denominator is intractable for all trajectory likelihoods of interest. Solutions to this problem mostly fall into two categories:

1. Sample from equation 2 numerically, using the samples to build up an estimate of the posterior distribution.
2. Construct a tractable approximation to 2.

Variational Bayesian methods fall into the latter category. The idea is to construct an approximation $q(\boldsymbol{\tau}, \boldsymbol{\theta}, \mathbf{Z}) \approx p\left(\boldsymbol{\tau}, \boldsymbol{\theta}, \mathbf{Z} | \mathbf{X}\right)$ that satisfies the following criteria:

$$q(\boldsymbol{\tau}, \boldsymbol{\theta}, \mathbf{Z}) = q(\mathbf{Z}) q(\boldsymbol{\tau}, \boldsymbol{\theta})$$
$$q(\boldsymbol{\tau}, \boldsymbol{\theta}, \mathbf{Z}) = \underset{\boldsymbol{\tau}, \boldsymbol{\theta}, \mathbf{Z}}{\operatorname{argmax}} L[q]$$

where $L[q]$ is the variational lower bound, also known as the evidence lower bound (ELBO):

$$L[q] = \int q(\boldsymbol{\tau}, \boldsymbol{\theta}, \mathbf{Z}) \log \left[\frac{p(\mathbf{X}, \boldsymbol{\tau}, \boldsymbol{\theta}, \mathbf{Z})}{q(\boldsymbol{\tau}, \boldsymbol{\theta}, \mathbf{Z})} \right] \mathrm{d}\boldsymbol{\tau} \, \mathrm{d}\boldsymbol{\theta} \, \mathrm{d}\mathbf{Z}$$

There are many resources that justify the use of $L[q]$ as the criterion for selecting q; here we simply remark that $L[q]$ is maximized when q exactly matches the true posterior distribution. Unlike criteria for maximization like the likelihood $p(\mathbf{X}|\boldsymbol{\tau}, \boldsymbol{\theta}, \mathbf{Z})$, maximization of $L[q]$ balances the ability of the model to describe the data against the complexity of the model. As a result, variational methods favor sparse mixture models with most elements of the state occupation vector $\boldsymbol{\tau}$ close to 0 when possible. This stands in sharp contrast to maximum likelihood methods, which tend to exploit all of the parameters of the model in order to explain the observed data.

State arrays rely on this property of variational methods by selecting a large K and fixing the state parameters $\boldsymbol{\theta}$ on a grid (essentially replacing the prior $p(\boldsymbol{\theta})$ with delta functions). For instance, if we are using a state array for RBME, our grid may span a range of diffusion coefficients and localization error variances.

Then $\boldsymbol{\theta}$ drops out from consideration, so our sole goal is to infer $\boldsymbol{\tau}$ and $\mathbf{Z}$.

Finally, we choose a Dirichlet distribution for the prior over $\boldsymbol{\tau}$:

$$p(\boldsymbol{\tau}) = \text{Dirichlet}\left(\frac{\alpha}{K}, ..., \frac{\alpha}{K}\right)$$

where α is the strength of the prior. α is also known as the number of pseudo-counts in the prior.

Under these conditions, the posterior distribution can be specified with the system of equations

$$\begin{aligned}
q(\boldsymbol{\tau}) &= \text{Dirichlet}\left(n_1, ..., n_K\right) \\
q(\mathbf{Z}) &= \prod_{i=1}^{N}\prod_{j=1}^{K} r_{ij}^{Z_{ij}} \\
r_{ij} &= \frac{\rho_{ij}}{\sum_{k=1}^{K} \rho_{ik}} \\
\rho_{ij} &= f_X\left(X_i|\theta_j\right) e^{\psi(n_j)} \\
n_j &= \frac{\alpha}{K} + \sum_{i=1}^{N} \frac{dL_i}{2} \mathbb{E}_{\mathbf{Z}\sim q(\mathbf{Z})}\left[Z_{ij}\right]
\end{aligned}$$

where $\psi(x) = \mathrm{d}\Gamma(x)/\mathrm{d}x$ is the digamma function. Notice that $\mathbb{E}\left[Z_{ij}\right] = r_{ij}$. So the scheme amounts to estimating the Dirichlet parameter $\mathbf{n}$ and the trajectory-state assignment probabilities $\mathbf{r}$.

We can infer $\mathbf{n}$ and $\mathbf{r}$ by alternating between them according to the following algorithm:

1. Initialize:
   (a) Evaluate $f_X(X_i|\theta_j)$ for each trajectory i and each state j. Represent these likelihoods as an N-by-K matrix $\mathbf{A}$.
   (b) Set $r_{ij}^{(0)} = A_{ij} / \sum_{k=1}^{K} A_{ik}$.
2. For each iteration $t = 1, 2, ...$:
   (a) For each state j, evaluate $n_j^{(t)} = \frac{\alpha}{K} + \sum_{i=1}^{N} \frac{dL_i}{2} r_{ij}^{(t-1)}$.

(b) Evaluate the matrix $\mathbf{r}^{(t)}$ such that

$$r_{ij}^{(t)} = \frac{A_{ij} e^{\psi(n_j^{(t)})}}{\sum_{k=1}^{K} A_{ik} e^{\psi(n_k^{(t)})}}$$

3. After convergence, the posterior distribution can be summarized with the posterior mean, which is

$$\mathbb{E}\left[\tau_j\right] = \frac{n_j}{\sum_{k=1}^{K} n_k}$$
$$\mathbb{E}\left[Z_{ij}\right] = r_{ij}$$

In addition, saSPT calculates a correction to the state occupations that accounts for the increased chance that fast-moving particles will diffuse out of the focal volume in a single frame interval.

2.4 Aggregated likelihood

An important naïve estimate for the state occupations can be obtained from the initialization $\mathbf{r}^{(0)}$ in the algorithm above. This is essentially the estimate for the trajectory-state assignments under the uniform prior over the state occupations. We can also estimate the naïve state occupations from this via

$$n_j^{(0)} = \frac{\alpha}{K} + \eta_j^{-1} \sum_{i=1}^{N} \frac{dL_i}{2} r_{ij}^{(0)}$$
$$\text{estimated occupation of state } j = \frac{n_j}{\sum_{k=1}^{K} n_k}$$

where η_j is a suitable correction factor for defocalization of state j. These "aggregated likelihoods" are highly useful for comparing variability in the diffusive profile for different files in the same dataset alongside the posterior mean.