The Lidke Lab Diffusion Estimator User's Guide

August 20, 2015

Contents

1 Introduction

1.1 The DEstimator Package

The DEstimator package is an object oriented classes built to operate under the Matlab environment. It is currently supported for Matlab versions 2014b through Matlab 2015b. The purpose of the DEstimator class is to organize and execute a variety of algorithms that perform operations based upon returning the likelihood of a given diffusion constant, D, given a trajectory described by: localized positions, localization errors, and frame time of associated localizations.

The DEstimator package is organized as a Matlab class, however, it is not necessary to use the package as a class–it can be used efficiently with the static functions, allowing the user full control over the likelihood call back functions.

1.2 DEstimator Algorithms

There are four main algorithms available for diffusion estimation in this package.

The recursive, Laplace, and Markov methods are all described in [CITE US ArXiv] and are available for use. The Laplace method can also decouple the maximum likelihood values of the positions from the localization variances.

We also include a slightly modified form of the work performed by Michalet and Berglund [Cite Michalet 2012].

1.3 DEstimator Main Functions

If the DEstimator package is used as a class, with trajectories loaded into the object, the main class methods are:

- **DEstimator** - The Constructor, creates an instance of the class.
- **initializeTrack** - Initializes a DEstimator object with an n-Dimensional trajectory.
- **LLH** - Evaluates the log likelihood at a sampled D.
- **LLHdim** - Evaluates the log likelihood of a specified dimension at a sampled D.
- **MLE** - Returns the $\hat{D}$, to numerical precision, which maximizes the likelihood.
- **MLEpositions** - returns the positions of X that maximize the likelihood of a given D. For the standard laplace method, this corresponds well with the motion averaged positions of the particle in simulations.

When individual functions of the DEstimator are called outside of the object environment, the static methods are:

- **computeEnsembleLLH** - Computes the log likelihood of a cell array of trajectories for a given sampling of D.
- **computeEnsembleMLE** - Returns the $\hat{D}$, to numerical precision, which maximizes the likelihood for a cell array of trajectories.
- **computeMLE** - Returns the $\hat{D}$, to numerical precision, which maximizes the likelihood.
- **computeLLH** - Evaluates the log likelihood at a sampled D.

1.4 DEstimator Testing Functions

The DEstimator package also contains some plotting and simulation static methods for testing the quality of the estimation algorithms:

- **simulate1D** - simulates a diffusing 1D particle with motion blur
- **simulate1DDiffusion** - simulates a diffusing 1D particle in the absence of all experimental sources of uncertainty

- **simulate1DObservationVar** - Adds experimental sources of uncertainty due to diffusion to a set of idealized coordinates.
- **generateObsVariance** - generates a vector of localization variances due to one of three distributions: uniform, gamma, or a constant (delta distribution).
- **compareLLHspeed** - compares the speed of the DST to the C++ implementations of the other three algorithms.
- **plotLLHspeed** - generates a figure from the results of **compareLLHspeed**.
- **compareLLHcomputations** - compares the likelihood curves of all available methods with the recursive method set as the benchmark.
- **compareMLEerror** - compares the MLE values of the recursiveC and the DST methods and calculates the associated risk from the squared log loss [cite Brown 1968] function.
- **plotMLEerror** - generates a figure from the results of **compareMLEerror**.

2 Installation

To install the DEstimator package, unzip all the contents into a folder on your computer. Open Matlab, access the folder in Matlab so you are in the proper directory, and run the file: startupDEstimator. That will put all of the necessary DEstimator files in the Matlab path.

3 Working with the DEstimator Class

The simplest way to use the DEstimator package is to treat it as a class and to load a trajectory into an object. The class based functionality was designed to minimize the required interface for analysis.

3.1 Creating a DEstimator object

To create an object, an instance of the DEstimator class, type:

```
est = DEstimator();
```

Where **est** can be any variable name that is acceptable by Matlab.

Example:

```
A = DEstimator();
```

3.2 Importing a Trajectory

Given an existing SPT trajectory, the inputs need to be formatted appropriately so that the DEstimator class can process the information. A trajectory consists of three matrices/vectors and a scalar exposure time. Globally, the units are not important, as long as they are consistent. So if position are in μm and times are in s, then the resulting D estimate will be in $\mu m^2/s$ if all inputs are consistent. Organize the trajectory so that its in the following format:

Obs = Observations (N time points x number of dimensions)

SE = Localization Standard Errors (N time points x number of dimensions). Note: Inputs are standard deviations of Observations.

Tvec = Times of each Observation set (N time points x 1)

exposureT = Scalar quantity that represents the exposure time of the camera sensor during a single image frame. If this input is not entered, the default value is the minimum time between observations, following the assumption that the minimum time spacing corresponds to a full frame integration.

algorithm = Text in single quotes that determines which estimation algorithm is used. If this input is not entered, the *recursiveC* algorithm is set as the estimator when the **initializeTrack** routine is called.

Once all the trajectory information is properly formatted, it can be loaded into the DEstimator class as:

```
success = est.initializeTrack(Obs, Tvec, SE, exposureT);
```

If success = 1, then the initialization process worked. If not, then the trajectory information was not parsed correctly.

3.3 Selecting an Estimation Algorithm

The default algorithm for the DEstimator class is *recursiveC*, however, there are a multitude of algorithms that can be used to compute the log likelihood. The list of algorithms can be returned by typing

```
DEstimator.Algorithms'
```

The algorithm used for estimations on a trajectory can be set by typing

```
est.algorithm = '[Algorithm Name]';
```

Example:

```
A.algorithm = 'LaplaceDirect';
```

3.4 Returning the Likelihood Distribution

The likelihood distribution of a vector of D values can be returned from a trajectory initialized DEstimator object as

```
llh = est.LLH(Dvec);
```

Where the output, **llh** is a vector of the same length as the input, **Dvec**, which is a vector of sampled D values for the likelihood distribution.

Also, for multi-dimensional trajectories, the log likelihood of a single dimension can also be returned with the following argument

```
llh = est.LLHdim(Dvec, dim);
```

where **dim** is a scalar specifying which dimension to evaluate.

3.5 Returning the Maximum Likelihood Estimate

The Maximum Likelihood Estimate (MLE) can be returned from a trajectory initialized DEstimator object as

```
[MLE llhD] = est.MLE();
```

Where the output **MLE** is the value of $\hat{D}$ that has the largest likelihood value for the input trajectory and **llhD** is the associated log likelihood value of the corresponding $\hat{D}$.

3.6 Positional Maximum Likelihoods

To get the MLE of the true particle positions given the free diffusion model, the argument is

```
[posMLE, posSE, llh] = est.MLEPositions(Dvec);
```

Where the input **Dvec** can be a vector or scalar of sampled D values. The corresponding outputs, **posMLE** and **posSE** are a tensors of positions and standard error, respectively, they have dimensions of observation length, trajectory dimension size, and Dvec size. The log likelihood values **llh** corresponds to each D value that is queried. Note that in simulated tests **posMLE** corresponds well with the motion blurred positions at each frame in the absence of localization variances.

4 Working with the DEstimator Functions

There are plenty of options for mobility analysis that are difficult to account for in an object oriented environment without sacrificing simplicity. To allow for versatility in mobility analysis, all of the DEstimator functions are provided as static methods.

4.1 Generating a Simulated Trajectory

A simulated 1-Dimensional trajectory with noise considerations can be generated from the static function

```
[Obs, Tvec, SE] = DEstimator.simulate1D
    (Dsim, N, SEin, dT, exposureT, varmethod);
```

with only scalar inputs. **Dsim** is the simulated Diffusion constant, **N** is the number of observations in the trajectory, **SEin** is the input of standard errors for the trajectory, **dT** is the time step between images, **exposureT** is the exposure time, and **varmethod** dictates the distribution the localization variances are chosen from. The last two inputs are optional with the default **exposureT = dT** and the default **varmethod** = 'gamma' from a set of three choices: 'gamma', 'constant', 'uniform'. The input **SEin** can be a scalar or vector, if its a scalar, a vector of **SE** are generated according to the distribution specified in **varmethod**, otherwise the vector **SEin = SE**. **Note:** The DEstimator trajectory functions do not generate intermittencies.

There are a few methods called inside of **simulate1D** that can be called separately if the user wants additional information in a simulation, such as the true positions. The following sub-subsections describe these methods with the same notation as used in this subsection.

4.1.1 Simulating Pure Diffusion

The true positions can be generated and simulated with

```
X = DEstimator.simulate1DDiffusion(N, Dsim, dT);
```

Where the output **X** are the true coordinates of a 1-D diffusive process.

4.1.2 Corrupting True Positions into Simulated Observations

A trajectory structure of **Obs**, **SE**, and **Tvec** can be generated from a set of true positions and scalar inputs with the command

```
[Obs, Tvec, SE] = DEstimator.simulate1DObservationVar
    (X, SEin, dT, exposureT, varmethod);
```

Where as in the **simulate1D** function, **exposureT** defaults to **dT** and **varmethod** defaults to 'gamma'.

4.1.3 Generating a Vector of Localization Variances

A specific vector of localization standard errors can be drawn from one of three distributions using the following command:

```
SE = DEstimator.generateObsStandardError(SEin, N, varmethod);
```

Where **varmethod** can either be 'gamma', 'constant', or 'uniform'. SEin must be a scalar in this instance. The 'gamma' method draws the localization standard errors from

$$\mathrm{SE} = \sigma \sim \mathrm{Gamma}(4, \mathrm{SEin}/4).$$

The 'uniform' method draws the localization variances from

$$\mathrm{SE} = \sigma \sim \mathrm{Uniform}(\frac{1}{2}\mathrm{SEin}, \frac{3}{2}\mathrm{SEin}).$$

The 'constant' distribution is actually a dirac delta distribution and sets all **SE = SEin**.

4.2 Working with Trajectories

Similar to the object oriented functions, the DEstimator static functions can calculate log likelihoods, MLEs, and position estimates.

A few of the inputs diverge from the object oriented approach described in Sec. 3.

Dvec is still a vector of sampled D.

Obs can only be a 1-dimensional vector of trajectory observations.

Tvec is a 1-dimensional vector of trajectory times.

SE can only be a 1-dimensional vector of localization standard errors.

exposureT is a scalar that represents the camera exposure time, if no value is chosen, it defaults to the minimum spacing between times seen in **Tvec**.

algorithm is the algorithm used as the estimator. 'recursiveC' is the default if no algorithm is chosen.

A convenient feature of free diffusion computations is that the log likelihood of several dimensions is equivalent to the sum of the log likelihoods calculated from each dimension.

4.2.1 Returning the Log Likelihood

Given a 1-D trajectory in the proper format, the log likelihood can be returned with the static command

```
llh = DEstimator.computeLLH(Dvec, Obs, Tvec, SE, exposureT, algorithm);
```

4.2.2 Returning the Diffusion MLE of a Trajectory

The $\hat{D}$ that maximizes the likelihood of a single 1-D trajectory can be returned with the command

```
[MLE, llhD] = DEstimator.computeMLE(Obs, Tvec, SE, exposureT, algorithm);
```

Where **MLE** = $\hat{D}$ and the associated log likelihood is **llhD**.

4.3 Working with Ensembles of Trajectories

In the static methods library, there are also a pair of functions designed specifically to handle ensembles of trajectories. The data formatting for trajectory ensembles is handled a bit differently. Rather than three vectors consisting of **Obs**, **Tvec**, and **SE**, a trajectory ensemble is put into a **Trajectories** cell array where each cell element is described as a matrix

```
Trajectories{ii} = [Obs, Tvec, SE]
```

where ii denotes the trajectory number. This allows for ensemble estimates of the likelihood of trajectories, where a trajectory of multiple dimensions can be treated as multiple trajectories of a single dimension.

4.3.1 Computing the Log Likelihood of an Ensemble of Trajectories

The log likelihood of an ensemble of trajectories can be calculated with the command

```
LLH = DEstimator.computeEnsembleLLH
    (Dvec, Trajectories, exposureT, algorithm);
```

4.3.2 Returning the Diffusion MLE of an Ensemble of Trajectories

The single $\hat{D}$ value that best describes an ensemble of trajectories can be calculated with the command

```
[MLE, llhD] = DEstimator.computeEnsembleMLE
    (Trajectories, exposureT, algorithm);
```

where **MLE** = $\hat{D}$ and the associated log likelihood is **llhD**.

5 Performance Testing of the DEstimator Algorithms

A few functions are imbedded in the DEstimator package to test the algorithms under a few metrics: speed, accuracy, and consistency.

5.1 Speed Testing the DEstimator Algorithms

The speed of each algorithm can be tested with the following command

```
results = DEstimator.compareLLHspeed(Nsizes, maxsize, trials, NDvals);
```

Where all of the inputs are scalars. **Nsizes** is the number of data points (trajectory lengths) to perform the speed test on, **maxsize** sets the longest trajectory length to test, **trials** determines how many trials are performed for each trajectory length, and **NDvals** determines the number of diffusion constants to test over. The speed test generates results for **Nsizes** number of trajectories logarithmal spaced from length 4 to the **maxsize**, sampled at various **NDvals** and repeated **ntrials**.

The output, **results** is a structure which can be plotted with the command

```
DEstimator.plotLLHspeed(results);
```

5.2 Accuracy Testing of the DEstimator Algorithms

The quality of each estimator can be tested with the following command

```
results = DEstimator.compareMLEerror
    (Nsizes, maxsize, Dsim, meanSE, dT, nTrials, varmethod);
```

Where all of the inputs are scalars. **Nsizes** is the number of data points (trajectory lengths) to perform the speed test on, **maxsize** sets the longest trajectory length to test, **Dsim** sets the simulated diffusion coefficient for all trajectories, **meanSE** is the mean value for the localization standard errors generated, **dT** is the time step (exposure time = **dT** for these simulations), **nTrials** is the number of trials at each trajectory length to estimate the risk from, and **varmethod** is the distribution to draw localization variances from.

As in the trajectory simulation methods, **varmethod** can only be set to 'gamma', 'constant', or 'uniform'.

If **nTrials** is set sufficiently high (¿ 100), then the resulting average losses makes a reasonable estimate of the risk. The risk for each estimator from the **results** structure can then be plotted with the command

```
DEstimator.plotMLEerror(results);
```

5.2.1 Algorithm Consistency Testing

The consistency of all the DEstimator algorithms can be checked with the command

```
DEstimator.compareLLHcomputations(ND, N, Dsim, meanSE, dT, exposureT);
```

Where all of the inputs are scalars. **ND** is the number of D values to return likelihood distributions from, **N** is the length of the trajectory to perform the comparison on, **Dsim** is the simulated diffusion constant, **meanSE** is the mean localization standard error which is drawn from a gamma distribution, **dT** is the time step for the simulated particle, and **exposureT** is the exposure time of each observation. If there is no input for **dT**, the default is 1 and if there is no input for **exposureT**, the default is **dT**.

This function automatically plots the results, providing the user with a visual comparison of all the algorithms.

6 Demo Script for 2-D Trajectory Analysis

This section will show how to generate a 2-D trajectory, return log likelihood values and then return the MLE $\hat{D}$ estimate. First, import or generate a trajectory. A trajectory can be generated by defining input parameters and typing in the keystrokes:

```
Dsim = 1; % Simulated diffusion constant
N = 100; % Number of observations in the trajectory
SEin = 0.5; % Mean standard error
dT = 1; % Time step per observation
exposureT = dT; % Exposure time
varmethod = 'gamma'; % Distribution of standard errors

% generate 2, 1D trajectories
[Ox T Sx] = DEstimator.simulate1D(Dsim, N, SEin, dT, exposureT, varmethod);
[Oy T Sy] = DEstimator.simulate1D(Dsim, N, SEin, dT, exposureT, varmethod);

% combine 2 trajectories into 1 2D trajectories
Obs = [Ox Oy];
SE = [Sx Sy];
```

A DEstimator object can then be created with the trajectory loaded in

```
A = DEstimator(Obs,T,SE,dT);
```

The associated log likelihood of several D values can be returned if the vector of D is defined

```
D_vec = linspace(1e-4,2,100);
LLH = A.LLH(D_vec);
```

The MLE, $\hat{D}$ of the diffusion constant given the trajectory can be estimated with MLE method

```
[MLE_D MLE_LLH] = A.MLE();
```

7 In Depth Components

There are many other functions imbedded in the DEstimator class, particularly, the function calls for each algorithm. However, the details are well documented inside the actual DEstimator class, so we omit their explicit description in this manual.