Tutorial for MATLAB reconstruction package

Contents

1. The package in brief

This package implements a maximum likelihood estimation (MLE) inference technique in order to fit diffusion (or Langevin) models to univariate time series data. The process of fitting a stochastic differential equation, including Langevin models, to data is commonly referred to as ‘system reconstruction’ in the literature (Siegert and Friedrich 2001, Rinn et al. 2016). For time series datasets with high and medium resolution, the package implements the Euler inference technique, which we refer to as ‘Euler reconstruction’. However, since Euler reconstruction cannot handle datasets with low resolution, the package also implements an MLE inference technique based on a refinement (Bakshi and Ju 2005) to a reconstruction approach developed by Aït-Sahalia (Aït-Sahalia 2002) for univariate data. This approach relies on Hermite expansion of densities, hence we call it ‘Hermite reconstruction’. There are two different modeling strategies the package can implement: parametric models and spline models. Spline modeling, offers an appealing framework, particularly when an appropriate functional form for the model is not straightforward. Splines are flexible structures that facilitate the recovery of unknown nonlinearities inherent in the data-generating system. Furthermore, since splines are linear functions in terms of model parameters, their use often leads to faster and more accurate results. The package is capable of reconstructing both ‘typical’ (single time series) and ‘replicate’ (several time series all believed to belong to the same data-generating system) datasets. Additionally, the package can analyze a portion of an extremely large typical or replicate dataset (and in the presence of missing values) sampled randomly across the entire data. Upon estimating model parameters, the package can calculate the corresponding uncertainty of the estimated parameters. We advocate for understanding the ideas and techniques by executing the code lines in this package step by, i.e., learning by doing.

The package is compatible with MATLAB 2022 and requires the following toolboxes: Curve Fitting Toolbox, Optimization Toolbox, Symbolic Math Toolbox, Econometrics Toolbox, and Signal Processing Toolbox. Additionally, it relies on the MATLAB package 'ARMASA' (Broersen, 2003), available for free download from the link https://nl.mathworks.com/matlabcentral/fileexchange/1330-armasa. Moreover, the code ‘armasel_s.m’ from the reference (Erkelens et al., 2013) is necessary. The authors have kindly permitted the inclusion of their code in our package. To ensure proper functionality, we have compiled both the ARMASA package and the ‘armasel_s.m’ code into a folder named ‘Burg’. Please add the link to this folder to your MATLAB working path.

2. Different data types and proper data formats

3. In this package, we distinguish between two types of data: 'typical' and 'replicate' data. Typical data refers to a single uninterrupted time series dataset, whereas replicate data consists of multiple separate time series datasets, all believed to originate from the same underlying data generating system. Typical data can be supplied as an array. On the other hand, replicate data should be supplied as cell arrays. Each cell should contain a single replicate dataset, following the same format as typical data. In other words, a replicate dataset should be organized as a cell array, with each cell containing a typical dataset. Both typical and replicate datasets can include ‘missing values’, which should be specified using ‘*NaN*’ notation. The package in brief

4. Two different modeling approaches

4.1 Parametric models

In this modeling framework, which we term *'parametric reconstruction'*, one specifies a diffusion model along with a parameter vector θ. A parametric diffusion model is represented by the stochastic differential equation:

$$dx = \mu(x;\theta)dt + \sigma(x;\theta)dW, \tag{1}$$

where $\mu(x;\theta)$ denotes the deterministic component of the system, known as the *'drift vector'*, and $\sigma(x;\theta)$ represents the stochastic component, known as the *'diffusion matrix'*. 'W' refers to a Wiener process, making the noise source dW Gaussian distributed and white (uncorrelated). It is worth noting that in some literature focusing on the equivalent

Fokker-Planck formulation of diffusion model (*1*), the function ½ $\sigma^2(x;\theta)$ is referred to as the diffusion function. The role of $\sigma(x;\theta)$ is to weigh the impact of noise source per state x measuring the noise intensity.

A diffusion model (*1*) is termed *'additive'* if the diffusion function $\sigma(x;\theta)$ is constant, meaning it does not vary with the state variable x (although it may depend on parameters θ). Otherwise, the diffusion model is termed *'multiplicative'*.

4.2 Spline models

Spline models are also parametric but they have a flexible form that can adapt to the shape of many nonlinear functions. Splines are an accurate tool for univariate data and fast to compute. To distinguish this modeling approach from parametric reconstruction we call it '*spline reconstruction'*.

Spline models are also parametric, but they offer a flexible form that can adjust to the shape of many nonlinear functions. In this modeling approach, the model parameters correspond to the values of the drift and diffusion functions over a relatively coarse mesh of the state space, known as the *'knot sequence'*. *What makes it convenient to work with splines is the fact that splines are linear functions in terms of parameters, even though they are non-linear in terms of state variables*. Unlike in parametric reconstruction, the user does not need to specify a model him/herself. Splines are particularly useful for univariate data and are computationally efficient. This package only considers spline modeling for univariate data. To differentiate this modeling approach from parametric reconstruction, we refer to it as *'spline reconstruction'*.

5. Two optimization solvers

We use two different optimization solvers in this package. The first solver is `‘fmincon’` which is a bult-in MATLAB solver. The second solver is the 'Grey wolf optimizer' (GWO) (Mirjalili et al. 2014), actually an improved GWO (Nadimi-Shahraki et al. 2021), abbreviated as `‘gwo’`, in this package. `fmincon` is a local solver but is a fast solver. `gwo`, on the other hand, is a global solver and is slower. We also utilize the MultiStart option to turn `fmincon` into a global solver. `fmincon` is our default solver.

6. A short description about the inputs and outputs of the main codes

The main code in the package is called `‘euler_reconstruction.m’`. Below, we explain its inputs and outputs here. Have a look here but the best way to learn is to run several examples after this section.

```
Res = euler_reconstruction(data,dt,'name',value,...)
```

`data`: Vector with a fixed time step

`dt`: The fixed time step between consecutive data points

name-value pairs

`'lb'`: Vector with the lower bounds of all parameters. For spline models the default is -10 for all knot values of mu and 0 for all knot values of sigma.

`'ub'`: Vector with the lower bounds of all parameters for spline models the default is 10 for all the knot values of mu and sigma)

`'L'`: Left boundary of the data (default is min(data))
`'R'`: Right boundary for the data (default is max(data))

Note: When working with small datasets, there may be fewer data points near the borders. This can adversely affect the quality of the fitted model, especially when using spline models (parametric models are not affected). To mitigate this issue, it is recommended to choose a relatively larger lower boundary and a smaller upper boundary for your data.

`'solver':` Optimization solver for the maximum-likelihood estimation problem. The solvers are `'fmincon'` and `‘gwo’` (default is `fmincon`). `‘gwo’` is a global solver but is slower.

`'gradient_fun':` The gradient vector of the objective function. All the optimization solvers can work without this but we recommend to use this option whenever applicable.

Note: This option is applicable only to parametric models (but is not applicable to 'Hermite reconstruction'). For nonlinear parametric models, it is recommended to utilize this option. This approach helps prevent the solver from becoming stuck at points that are not even local minima (stagnation).

`'useparallel':` Use parallel computing (default is false)

`'search_agents':` Number of searching agents (default is 5)

`'maxiter':` Maximum number of iterations (default is `'realmax'` which, in practice, means infinity)

`'nknots':` Is a two-element vector where the first (second) element specifies the number of knots you want to allocate for `mu` (`sigma`). For additive noise use `[n 1]` which means you use `n` knots for `mu` and a single knot for `sigma` (default is `[8 8]`).

`'knots':` The values of the knots (alternative to `nknots`).

`'spline':` a two-element string which specifies the types of splines for mu and sigma (the default is `'CC'`). Spline types for mu and sigma are as follow

`'L'` = linear interpolation (i.e., a straight line)
`'C'` = cubic spline interpolation
`'Q'` = quadratic spline interpolation
`'P'` = `pchip` spline interpolation (`pchip` respects the monotonicity in data)
`'SCS'` = cubic smoothing spline
`'Approximate'` = `'SCS'` but uses `'L'` for fast fitting

(For instance, `'LL'` means that you want to specify linear interpolation `'L'` for both mu and sigma. `'CL'` means that you want to specify cubic spline `'C'` for `mu` but a linear spline `'L'` for `sigma`. Likewise, '`SCSL'` means `'SCS'` for `mu` and `'L'` for `sigma`).

The following name -value pairs are only suitable for parametric models

`'mu'`: parametric function handle for mu.

`'sigma'`: parametric function handle for sigma (if it is empty then an additive sigma (i.e., constant) is considered)

Note: mu and sigma must be vectorized. For instance, `mu = @(x,par)par(1)*x^3+par(2)*x` is not suitable, rather `mu = @(x,par)par(1).*x.^3+par(2).*x` is appropriate.

`'npars'`: (optional) number of parameters.

`'gradient_fun'`: handle to gradient function (generated with `'eulergrad'`). If you want the code to calculate the gradient use this option (then you might get a more accurate result).

The optimizing dialog screen

While the package is solving the MLE problem, a dialog screen appears, providing updates on the optimization progress at each iteration. This feature is particularly valuable when applying Hermite reconstruction, allowing users to monitor the outcomes closely. If the results show gradual improvements, users can safely stop the code by pressing the 'stop' button.

A simple plotting option

After you are done with the command `'res=euler_reconstruction(data,dt,'name',value,... )'` you can enjoy a nice graphical interface using the command `'plot_results(res)'`.

7. Check three requirements in real datasets in advance

7.1 Data stationarity

In order to fit a diffusion model (*1*) to data, it is essential for the data to be stationary, at least in a weak sense. In simpler terms, stationarity implies that the statistical properties of the system, and hence the dataset under consideration, remain invariant over time. Weak stationarity within a fixed time window entails that the mean and variance of the data remain constant, and that the autocorrelation function depends solely on the time lag rather than on the initial and final times within the specified window. However, if stationarity is violated across the entire dataset, it may be possible to identify shorter, (possibly overlapping) time windows during which the data exhibit stationarity (see **Example 10** in subsection 10.3). Reconstruction can then be performed separately for each of these windows. The stationarity of the data can be assessed using the Augmented Dickey-Fuller test (ADF test) (Dickey and Fuller 1979). We conducted the ADF test on all real datasets in this tutorial. It is worth noting that simulated data are inherently stationary because model (*1*) is stationary.

As an example, let's apply the ADF test to the `'OUdata1D.mat'` dataset as below

```
S = load('OUdata1D.mat');
data = S.data;
[~,~,~,~,reg] = adftest(data,'model','ARD','lags',0:20);  % the input 0:20 is the
number of lags we try in fitting an autoregressive model to data.
[~,lagndx] = min([reg(:).BIC]);  % this tells us how many lags we need (lagndx is 1)
[h, Pvalue,~]=adftest(data,'model','ARD','lags',lagndx);h
```

```
1
```

In this test we apply ADF test at least twice. First, we consider an array of lags and apply ADF test to see how many autoregressive lags our data needs. In the above example we considered the lags `0:20` and the quantity `lagndx` tells us the required number of lags (which corresponds with the lowest `BIC`) we need which is `1` (if we get `20` then clearly, we should repeat the command `adftest(data,'model','ARD','lags',0:20)` with a bigger array of lags). Next, we apply the ADF test again and the output `h` is the test result. If `h = 1` (as is the case above and this was expected) then the dataset is stationary, otherwise it is non-stationary. As for another example with real data type the following

We should apply the ADF test at least twice. Firstly, we consider an array of lags and apply the ADF test to determine the optimal number of autoregressive lags required for our data. In this example, we considered lags ranging from `0` to `20`, and the quantity `lagndx` indicates the number of lags corresponding to the lowest Bayesian Information Criterion (`BIC`). For instance, if `lagndx` returns `1`, it suggests that only one lag is required. However, if `lagndx` returns `20`, it indicates the need to repeat the ADF test with a larger array of lags. Subsequently, we perform the ADF test again, and the output `h` represents the test result. A value of `h = 1` indicates that the dataset is stationary, as observed in the example above. Conversely, if `h` is not equal to 1, it signifies that the dataset is non-stationary. For another example, using real data, we try the following command lines

```
data = readmatrix('NGRIP20.csv');
[~,~,~,~,reg] = adftest(data,'model','ARD','lags',0:20);  % the input 0:20 is the
number of lags we try in fitting an autoregressive model to data.
[~,lagndx] = min([reg(:).BIC]);  % this tells us how many lags we need (lagndx is 5)
[h, Pvalue,~]=adftest(data,'model','ARD','lags',lagndx);h
```

```
1
```

This confirms the stationarity of the second dataset. All datasets in this tutorial are stationary.

7.2 Data Markovicity

First of all, the MATLAB package called 'ARMASA' (Broersen 2003) is needed for this section. This package can be freely downloaded from the following link

https://nl.mathworks.com/matlabcentral/fileexchange/1330-armasa

Furthermore, for the analysis of replicate data, another code called 'armasel_s.m' from the reference (Erkelens et al. 2013) is needed. We have contacted the authors, and they have graciously allowed us to include their code in our package. We have created a folder called 'Burg' which contains both the ARMASA package and the code 'armasel_s.m'. You should add the link of this folder to your MATLAB working path.

Reconstructing real datasets presents unique challenges, notably due to the correlation of noise at very small scales, a phenomenon highlighted by Einstein in his seminal work on Brownian motion (Einstein 1905). Diffusion models in (*1*), on the other hand, are Markov models (In short, the Markov property dictates that the future state of a system, given its present state, is independent of the entire history of past states). Consequently, the reconstruction process must adhere to a specific time scale, known as the '*Markov-Einstein' (ME) time scale* (Friedrich et al. 2011), ensuring the fulfillment of the Markov property. This implies that if the ME time scale equals 1, the entire dataset is Markov and can be used directly. However, for an ME time scale of 2, only every second data point should be included in the analysis, and so forth. It is important to note that the Markov property holds at any time scale larger than the ME time scale, allowing for reconstructions at these higher scales as well. Determining the ME time scale, however, is far from straightforward. Traditional methods for estimating the ME time scale often involve binning and require extensive data (Friedrich et al. 2011), leading to results that may vary significantly with the chosen bin size and potentially introduce bias in smaller datasets. To address these challenges, we propose a more streamlined and data-efficient method. This approach involves fitting an autoregressive (AR) model to the data and examining the order of the fitted AR model. Specifically, if an AR(1) model emerges as the optimal fit, this strongly suggests an ME time scale of 1, indicating that the dataset in question is Markov. Similarly, an AR(p) model suggests an ME time scale of p. Here, we assess the ME time scale for a few datasets. As first example, type the following commands

```
S = load('OUdata1D.mat');
data = S.data;
order = 10;  % order is the maximum AR order being considered (should be long enough)
AR = ME_TimeScale(data,order)
```

and you get

```
AR = [1.0000   -0.9899]
```

This indicates that an AR(1) model provides the best fit for this dataset (note that the number of elements in the above `AR` vector after the first element, which is always 1, corresponds to the order of the fitted AR process.) Therefore, the ME time scale is 1, confirming the Markovian nature of this dataset. This was expected, since this dataset is simulated. Let's now examine a real dataset

```
data = readmatrix('BGA_stdlevel_2011.csv');
order = 10;
AR = ME_TimeScale(data,order)
```

and you get

```
AR = [1.0000   -0.6261   -0.2177   -0.0840   -0.0339   -0.0120   -0.0068   -0.0009
-0.0080    0.0005   -0.0089]
```

This suggests that the dataset exhibits long-range correlations. However, the magnitudes of the `AR` coefficients beyond the third element (`-0.2177`) or the fourth element (`-0.0839`) are small and we can safely assume that the AR order is `p = 3` or even `p = 2`. Consequently, the ME time scale is estimated to be 3. This indicates that the dataset does not adhere to the Markov property. However, by considering every third data point (i.e., `data(1:3:end)`), the resulting rarified dataset satisfies the Markov property. Further rarefication, such as considering every higher order of data point, also results in a Markovian dataset. Therefore, for reconstruction purposes, it is essential to apply the reconstruction algorithms to the rarified datasets rather than the original one. Finally, let's proceed to examine the ME time scale for another real dataset in this tutorial

```
data = readmatrix('NGRIP20.csv');
order = 10;
AR = ME_TimeScale(data,order)
```

and you get

```
AR = [1.0000   -0.4879   -0.2198   -0.1231   -0.0541   -0.0030   -0.0186   -0.0290
-0.0159   -0.0061   -0.0262]
```

Roughly speaking, the ME order is 2 or 3. However, since this is a very small dataset and that its resolution is also low we considered every other data points, i.e., `data(1:2:end)`.

7.3 Data resolution: a key data feature in this study

Prior to embarking on system reconstruction, it is crucial to estimate the resolution of the data. This estimation provides a rough categorization of the data into ‘high-resolution’, ‘medium-resolution’ or, ‘low-resolution’ categories, which is essential for selecting an appropriate reconstruction algorithm. To achieve this, we need to estimate a quantity called ‘*relaxation time scale’* of the yet unknown data-generating system. In a system with `N` state variables, there exist `N` time scales, and determining these scales is a challenging task (see Appendix A). Typically, the time scales of a nonlinear system are estimated using a linear Ornstein-Uhlenbeck (OU) system, a process that involves numerous approximations and simplifications. However, it is not necessary to accurately estimate the time scales; rather, having a general ‘*feel’* for them is sufficient. Building on insights from the previous section, we recognize that reconstruction for real datasets should be conducted on a rarified sample of data that exhibits Markov property. Consequently, when dealing with real data, it is important to estimate the relaxation time for this specific sample of data, rather than the entire dataset.

To determine the resolution of the data, use the command ‘`RelaxationTime`’ (expressed in terms of number of time steps, i.e., sampling time dt). We give a ‘loose’ but practical convention for categorizing data resolution into three categories: high, medium, and low. This convention is derived from extensive experience with numerous datasets rather than a rigorous mathematical foundation. It is particularly useful for determining when to use Euler reconstruction versus Hermite reconstruction (see Section 10). Specifically, ‘*a dataset with a relaxation time in the interval [1,50] is considered to have low resolution, while a relaxation time in the interval [50 100] indicates medium resolution and a relaxation time greater than 100 is indicative of high-resolution’*. For a bivariate system, as mentioned earlier, we have two time scales. By comparing these time scales, we can decide on the category to which our data belongs. As an example, let’s try finding the relaxation time for the dataset in **Example 1** in Section 9. Type the following commands

8. The package in brief

9. The package in brief

10. The package in brief

11. The package in brief

```
S = load('OUdata1D.mat');
data = S.data;
R = RelaxationTime(data); R
98.1858 (number of time steps)
```

As we expected (since this dataset was generated from the OU model $dx = -x\,dt + dW$, with a simulation time step dt of 0.01. In theory, R should be 100 time units. However, due to the finite size of the data, we obtained R~8.18). Now, we assess the ME time scale for an ecological dataset. Type the following commands

```
data = readmatrix('BGA_stdlevel_2011.csv');
data = data(1:3:end);  % Important: We learned from previous section that ME time
scale in this dataset is 3. So, we must consider every third data point
R = RelaxationTime(data); R
```

```
273.7843 (number of time steps)
```

It is important to pay attention to the fact that this real dataset is not Markov. However, the rarified sample of this dataset, obtained by considering every third data point, i.e., data(1:3:end), is Markov. Therefore, when estimating the relaxation time, it is essential to base it on this sample rather than the entire dataset. Despite being rarified, this sample maintains a high resolution. As for another real dataset we examine the ME time scale of a univariate ice-core dataset. Type the following

```
data = readmatrix('NGRIP20.csv');
data = data(1:2:end);  % Important: We learned from previous section that ME time
scale in this dataset is 2. So, we must consider every other data points
R = RelaxationTime(data); R
```

```
39.6383 (number of time steps)
```

Which places the rarified sample of this dataset, obtained by considering every other data point, in the category of low-resolution.

Finally, we examine the relaxation time of a replicate dataset as below

```
S = load('MayData1D_Replicate.mat');
data = S.data;
R = RelaxationTime(data); R
```

```
219.5342 (number of time steps)
```

And this puts this rarified sample in the category of low-resolution. The computational burden in the second phase of reconstruction (called Hermite reconstruction, see section 10) increases as we increase either J or K (these are parameters of the Hermite reconstruction). A small relaxation time indicates the need for choosing large values of K (for a fixed J) for the estimation procedure to work efficiently. Conversely, a large relaxation time means that highly accurate result can be obtained with small K (for a fixed J). Note that here 'small' relaxation time refers to small values, typically close to 1 from above, while 'large' indicates values significantly larger than 1. **Table 1** summarizes the data requirements for all the datasets in this tutorial.

Datasets	Real or simulated?	ME time scale	Relaxation time scale (s)	Category
OUdata1D.mat	Simulated	1	98.18	(almost) high-resolution
MayData1D.mat	Simulated	1	3641.8	high-resolution
MayData1D_Replicate.mat	Simulated	1	219.5342	high-resolution
BGA_stdlevel_2011.csv	Real (ecology)	2 or 3	273.7843	high-resolution
OUdata1D.mat (Every 100th data points)	Simulated	1	1.0073	Extremely low-resolution
MayData1D.mat (Every 300th data points)	Simulated	1	12.79	low-resolution

NGRIP20.csv (We analyzed every other data point)	Real (ice-core)	2 or 3	15.0951	low-resolution

Table 1. A summary of data requirements for all the datasets in this tutorial.

12. Simulating data from parametric and spline models

The command for the simulations is `'simulate'`. In order to generate a dataset from a parametric model, you need to specify the parameters: the model type `ModelType` (i.e.,`'parametric'`), a lower bound `L` for data (if empty, the code considers it to be $-\infty$), an upper bound `R` for data (if empty, the code considers it to be ∞), the drift vector `mu` (which should be a function handle), the diffusion matrix `sigma` (which should also be a function handle), the initial state `x0`, time step `dt` (which should be small, relative to the scale of the problem), and the number of data points `T`. As a first example, consider the one-dimensional Ornstein-Uhlenbeck (OU) model $dx = \mu x\,dt + \sigma dW$ with parameters $\mu = -1$ and $\sigma = 1$. The following command lines generate a dataset from OU model, starting from $x_0 = 0$, with the time step of $dt = 0.01$ and $T = 10^5$ data points (see Figure2, left panel, for an illustration)

```
ModelType = 'parametric';
L = [];  % lower bound on data is -infinity
R = [];  % upper bound on data is infinity
a = -1;s = 1;
mu = @(x)a.*x;  % drift function
sigma = @(x)s+0.*x;  % diffusion function
dt = 0.01; % time step of Euler-Maruyama integration
x0 = 0;  % initial state
T=10^5;    % simulation length
x=simulate(ModelType,L,R,mu,sigma,dt,x0,T);
```

as you see the lower bound `L` and the upper bound `R` are empty. Therefore, the package, by default, considers a lower bound of $-\infty$ and an upper bound of ∞.

Finally, let's simulate a dataset from a spline model. Note that in this package spline modeling is only possible for one-dimensional models. To generate a dataset from a spline model, you need to define the parameters: model type `ModelType` (i.e., `'spline'`), `SplineType` (with `'CC'` being the default and often used), `knots` (a rather sparse mesh across the state space), a vector of parameters (which are the corresponding values of drift and diffusion functions at knots), a lower bound `L`, an upper bound `R`, a time step `dt`, and the number of data points `T`. For details on these inputs see section 5. The following command lines generate data for a spline model (which is reconstructed via a spline reconstruction in `section 9.2`, **Example 4**)

```
ModelType = 'spline';
SplineType = 'CC';
L = -4;
R = 4;
knots = linspace(L, R, 8);
par =[4.8855    1.1983    1.1006    0.24633   -0.10136   -0.69903   -1.3182    -6.4891 ...
0.84428    1.0242    1.0017    0.98554    1.0031     0.97949    1.0185     0.79658]; % this
%is a parameter vector estimated following a spline reconstruction in subsection 9.1, Example 3
dt = 0.01; % time step of Euler-Maruyama integration
x0 = 0;  % initial state
T=10^5;    % simulation length
x=simulate(ModelType, SplineType, par, L, R, knots, dt, x0, T);
```

13. Data standardization

To help the reconstruction procedure, it is better to standardize the data by subtracting the mean and dividing by the standard deviation. This step is especially important when the range of data is large. Standardization helps confine the search region into smaller and more manageable searching spaces, reducing the risk of numerical instabilities. Simulated data and one real dataset are not standardized in this tutorial since their range is not extensive. For an example of data standardization see Section **Error! Reference source not found.Error! Reference source not found.** (**Example 10Example 10** and **Figure 6**). If standardization is applied to data, then the reconstruction procedure follows these steps: data standardization, performing the reconstruction on the standardized data, and then back-transforming the results to the original magnitude and scale of the original data.

Consider the diffusion model (**Error! Reference source not found.**), i.e., $dx = \mu(x;\theta)dt + \sigma(x;\theta)dW$ and the standardization $z = (x - m_{\mathrm{d}})/s_{\mathrm{d}}$ where m_{d} and s_{d} are the mean and standard deviation of data. If the diffusion model $dz = \mu_z(z)dt + \sigma_z(z)dW$ describes the dynamics of the transformed process z, then the corresponding diffusion model for the original process x is as follows:

$$dx = s_d\mu_z\big((x - m_d)/s_d;\theta\big)dt + s_d\sigma_z\big((x - m_d)/s_d;\theta\big)dW, \tag{2}$$

Otherwise, it is a normal product. If either of the drift vector or diffusion matrix is linear in the parameters, then all we need to do is to replace the estimated vector of parameters θ in (2) with $\mathrm{s_d}.\theta$ and remove the factor $\mathrm{s_d}$. For instance, if both μ_z and σ_z are linear in terms of the parameter vector θ, then (2) will be simplified to

$$dx = \mu_z\big((x - m_{\mathrm{d}})/s_{\mathrm{d}}; s_{\mathrm{d}}.\theta\big)dt + \sigma_z\big((x - m_{\mathrm{d}})/s_{\mathrm{d}}; s_{\mathrm{d}}.\theta\big)dW.$$

Except for `pchip` splines, all the splines in this package are linear functions of parameters (though not of state variable). Even `pchip` splines are close to being linear. This reflects the ease of working with splines.

14. Euler reconstruction

14.1 Reconstructing a dataset simulated from a linear model

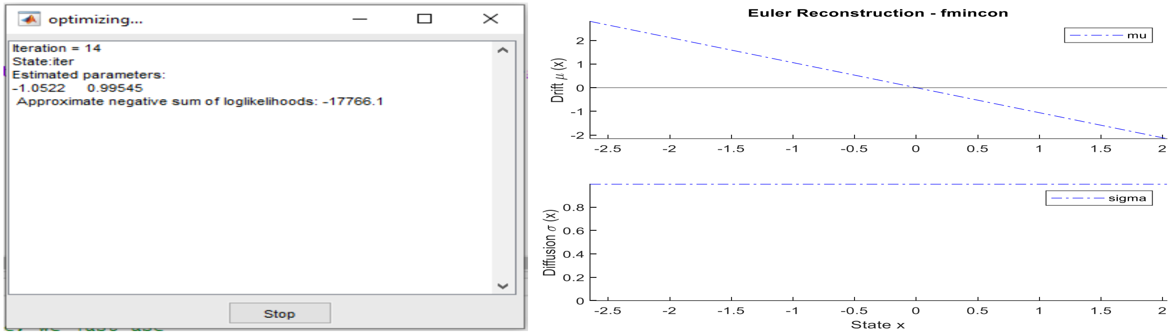
Example 1. In the first case study we apply the parametric and spline reconstruction techniques to a dataset being simulated from the OU process $dx = \mu x\, dt + \sigma dW$ with parameters $\mu = -1$ and $\sigma = 1$. In this dataset we used the time step of `dt = 0.01` and number of data points are `T=10`^{6} although here we only use the first 20000 data points (See **Figure 2**, left panel. See also **Example 10** where we use a very sparse sample of the entire dataset). Type the following commands

```
S = load('OUdata1D.mat');
data = S.data;  %load the data
data = data(1:20000); %This is a big timeseries with 10^6 data points. Here, we just
%use its first 20000 data points
dt = 0.01;
mu = @(x,par)par(1).*x;
sigma = @(x,par)par(2);
result1 = euler_reconstruction(data, dt, 'mu', mu, 'sigma', sigma,  ...
'gradient_fun', eulergrad(mu, sigma), 'lb', [-200 eps], 'ub', [200 200]);
```

And, you get

```
Estimated parameters :
-1.0586     0.99531
- sum of log-likelihoods : -17766.0856
```

While the package is solving the problem, a dialog screen appears on your screen (see **Figure 1**, left panel). This dialog screen is particularly helpful when the reconstruction process is slow or shows gradual improvement, such as when dealing with large or low-resolution datasets that require Hermite reconstruction. By monitoring the dialog screen, you can observe any slight improvements in the reconstruction process. If you notice minimal improvement, you can terminate the process by pressing the `'stop'` button (of course, it is not the case with this small dataset). To assess the progress of the reconstruction, pay attention to the objective value displayed on the dialog screen, which represents the negative sum of log-likelihoods. A lower objective value indicates a better fit. If the objective value stops declining or decreases very slowly, you can safely terminate the code.



```
Iteration = 14
State:iter
Estimated parameters:
-1.0522    0.99545
 Approximate negative sum of loglikelihoods: -17766.1
```

Figure 1. The left panel displays a dialog screen that appears during the package's execution, providing parameter updates during the optimization process. Additionally, the right panel showcases a graphical representation of a parametric model (**Example 1**) utilized for reconstructing the OU model.

We have only used the first 20000 data points of this large dataset with 10^6 data points (we will need the entire data in the later sections). The model is parametric as we have defined the drift `mu(x)` and diffusion `sigma(x)` functions using function handles. The model consists of two parameters: one for the drift function and one for the diffusion function. We have specified a vector of lower bounds as `[-200 eps]`, where the lower bound for the first parameter is -200 and the lower bound for the second parameter is `eps`, serving as an 'infinitesimal'. It is important to note that the diffusion function must remain positive. While the code generally cannot check for the positivity of the diffusion function in parametric models, for this simple additive model, the code will alert you if you overlook this requirement. However, for more complex multiplicative noise models, you must verify this by yourself (For instance, if `sigma(x)= x.^2+1+a` then `a>-1` should be fulfilled for the `sigma(x)` to remain positive and you can easily check this by plotting `sigma(x)` as a function of state `x` and parameter `a`). Additionally, we have defined a vector of upper bounds as `[-200 200]`, indicating that both parameters are bounded by `200` from above. To visualize the results, use the command `plot_results(result1)`.

Example 2. Let's now try a bit different parametric model here. Type the following commands

```
S = load('OUdata1D.mat');
data = S.data;
data = data(1:20000);
data(1:100:end) = nan;  %this is to show you that the package works in the presence
of NANs
dt = 0.01;
mu = @(x,par)par(1).*x+par(2).*x.^2;sigma = @(x,par)par(3); %here, we have 3
%parameters
result2 = euler_reconstruction(data, dt, 'mu', mu, 'sigma', sigma,  ...
'gradient_fun', eulergrad(mu, sigma), 'lb', [-200 -200 eps], 'ub', [200 200 200]);
```

and you get

```
Estimated parameters :
-1.0735    -0.036727     0.99536
- sum of log-likelihoods) : -17410.5382
- sum of log-likelihoods : -17766.2684
```

In this example, we have introduced `NaN` values into every 100th data point to demonstrate the package's capability to handle missing data. Additionally, we have augmented the former drift model with an extra quadratic term, `par(2)*x.^2`, to assess if the package recognizes unnecessary terms. The second parameter is estimated to be `-0.036727`. Notably, the second parameter estimate remains small, as expected. However, when using larger data portions, the estimate for the second parameter tends to decrease. Now, which model provides a better fit? `result1` or `result2`? To answer this question, compare the objective function values, i.e., the negative sum of log-likelihoods. A lower objective value indicates a better fit. Surprisingly, `result2` exhibits a slightly smaller objective value, suggesting it is a better fit. This outcome may seem counterintuitive, but it is expected. With a finite dataset length, models with more parameters tend to yield better fits. However, as the dataset size increases, the discrepancy between the models diminishes, and the estimates of additional parameters converge to zero

Example 3. Now, let’s try a spline model for our data. Type the following commands

```
S = load('OUdata1D.mat');
data = S.data;  %load the data
data = data(1:20000);
dt = 0.01;
L = -2;
R = 1.8; %since we have a ‘spline’ model it is better to shrink the state space
data(data<L | data>R) = nan;  %This is VERY important: In spline modeling if you
consider a smaller range for your data then you must assign ‘nan’ to those few data
points falling outside this range.
mu = 8;
sigma = 8;  %since mu and sigma are numbers this means that we want to consider
%spline modeling with 8 knots for mu and 8 knots for sigma
result3 = euler_reconstruction(data, dt, 'nKnots', [nmu nsigma], 'spline', 'CC', 'L',
...
L, 'R', R, 'lb', [zeros(1, mu) - 10, zeros(1, sigma)+eps], 'ub', zeros(1, mu + sigma)
+ 10, 'solver', 'fmincon'); %we have 8+8 parameters, so, ‘lb’ and ‘ub’ should have 16
%elements. The vector of lower bounds ‘lb’ has 8 lower bounds for mu (which are -10) and 8
%lower bounds for sigma (which are eps, i.e., infinitesimal). All 16 elements of ‘ub’ are 10
```

and, you get

```
Estimated parameters:
4.8855    1.1983   1.1006   0.24633  -0.10136  -0.69903  -1.3182   -6.4891
0.84428   1.0242   1.0017   0.98554   1.0031    0.97949   1.0185    0.79658
- sum of log-likelihoods): -17730.1649
```

In this example, we have specified a spline model, unlike the previous examples. When using a spline model, the drift `mu` and `sigma` functions should be numeric values only. For instance, specifying `mu=8` indicates a spline model for the drift function with 8 equidistant knots across the state space (the same applies to `sigma=8`). By default, the state space is set to `[min(data) max(data)]`, but it is recommended to narrow this range by specifying a larger lower limit `L` and a smaller upper limit `R` for the data. This adjustment is beneficial because there are typically very few data points near the data borders, which can adversely affect the accuracy of the `mu` and `sigma` functions near data borders (i.e., the first and last knots) (see **Figure 2**, right panel). For this dataset containing 20000 data points, we have chosen the range `[-2 1.8]`, as only 0.000028% of the data fall outside this range (see **Figure 2**, left panel). *It is crucial to note that when utilizing spline modeling and opting for a smaller range for your data, any data point falling outside this reduced range must be assigned `NaN`.* This ensures that the spline model accurately reflects the specified data range and 'respects' the order of data in the smaller dataset. To visualize the estimated results and the true model, use the following commands (see **Figure 2**, right panel).

```
mu = @(x,par)par(1).*x;
sigma = @(x,par)par(2)+0.*x; %this is true model
par = zeros(2,1);par(1) = -1;par(2) = 1; %true model parameters
xplot = linspace(L,R,1000); % a dense mesh across the considered range
plot_results(result3,xplot,mu(xplot,par),sigma(xplot,par));
```

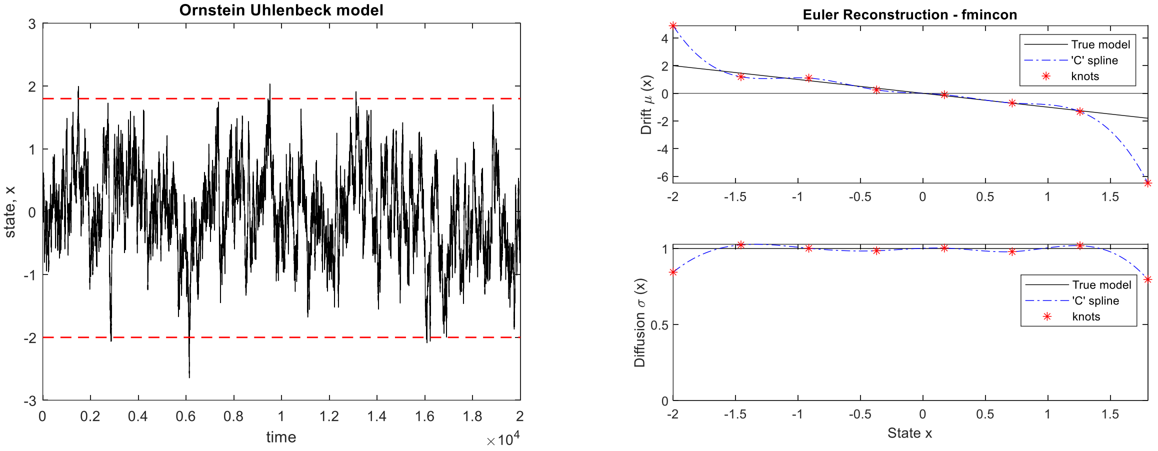


Figure 2. (Left panel) the first 20000 part of a dataset along with the reduced range being considered, highlighting the vast majority of data points within the specified range. The dataset is generated from the OU process $dx = \mu x\, dt + \sigma dW$ with parameters $\mu = -1$ and $\sigma = 1$, with a time step of `dt = 0.01` and number of data points are `T=10`^{6}. (Right panel) a graphical representation showcases a spline model featuring `mu = 8` knots (indicated by red stars) for the drift function and `sigma = 8` knots for the diffusion function. These functions are represented by dot-dashed blue curves, while the true model is depicted by solid black curves. The data are as in **Figure 1**.

14.2 Reconstructing a dataset simulated from a nonlinear model

Example 4. In the second case study we apply the parametric and spline reconstruction techniques to a dataset being simulated from the following stochastic version of overgrazed model of May(May 1977) with additive noise

$$dx = \left\{ rx\left(1 - \frac{x}{K}\right) - \frac{\gamma x^2}{x^2 + a^2} \right\} dt + \sigma\, dW,$$

where the model parameters are $r = 1.01, K = 10, \gamma = 2.75, a = 1.6,\ \sigma = 0.4$. We have simulated a dataset containing $3 * 10^5$ data points with time step $dt = 0.01$ (**Figure 3**, left panel). We fit a parametric model to the first third of this dataset. Type the following

```
S = load(‘MayData1D.mat’);
data = S.data;  %load the data
data = data(1:100000); %We only use the first third of the dataset
dt = 0.01;
mu = @(x,par)par(1).*x.*(1-x./par(2))-par(3).*x.^2./(x.^2+par(4).^2);
sigma = @(x,par)par(5);
result4 = euler_reconstruction(data, dt, 'mu', mu, 'sigma', sigma, 'gradient_fun',
eulergrad(mu, sigma), ...
    'lb', zeros(1,5)+eps, 'ub', 15.*ones(1,5),'useparallel',true,'search_agents', 5);
```

and, you get

```
Estimated parameters :
1.1805      9.8416      3.1242      1.5279     0.39972
- sum of log-likelihoods) : -180062.6377
```

This is a nonlinear model, making it more challenging to solve the underlying optimization due to the presence of multiple local minima. When dealing with nonlinear problems, we recommend using the name value pairs `'gradient_fun'` to mitigate the risk of stagnation. Stagnation occurs when the optimization process gets stuck in a solution, which may not even be a local minimum (for instance, when the objective function is very flat). In such situations the code may continue for a long time, requiring manual intervention to stop and restart. To aid in identifying stagnation, dialog box is essential, allowing users to monitor the optimization process and stop it whenever it progresses slowly. In this example, the searching region is chosen to be a cube in the positive orthant with sides of `15`. Increasing the size of searching region raises the risk of stagnation. How can we determine an appropriate search region a priori? Here are several considerations:

1. Utilize the `fmincon` solver, which is the default solver and is fast. Even if `fmincon` fails to converge to the true solution, it provides valuable insights into the approximate search region to explore.
2. Consider using our global solver `gwo`, although it may not be as fast. Initially, apply `gwo` to a large search region with a significant number of `search_agents` (e.g., 500) for a brief period to identify a smaller, more appropriate search region. This is not intended to find the true solution, but rather to gain insights into the search region for exploration later using `fmincon`.
3. Opting for spline modeling simplifies and facilitates the optimization problem, as splines are linear functions of parameters, despite being non-linear in state variables. Standardizing the data (subtracting the mean and dividing by the standard deviation) enables the selection of narrower search regions for the parameters of `mu` and `sigma`. We, recommend to use a searching region of `[-10 10]` for parameters of `mu` and `[eps 10]` for parameters of `sigma`, respectively (see **Example 3**). If it turns out that this search region is still small, you can simply choose a larger time step `dt`. It is important not to confuse this with the time step used for simulations, which should indeed be small. When performing the reconstruction, `dt` acts as a 'scale' parameter, so the specific number chosen is not important. For the simulated data, we chose the same time step `dt`, as in the simulations for proof of concept, even though it wasn't necessary to do so. Since splines are linear in terms of parameters, multiplying `dt` by a factor `k` will divide the drift parameters by `k` and the diffusion parameters by the square root of `k`. In technical terms, in a diffusion model (**Error! Reference source not found.**), i.e., the model $dx = \mu(x)dt + \sigma(x)dW_t$ under the change of time scale $\tau = kt$ the diffusion model becomes $dx = 1/k\ \mu(x)d\tau + 1/\sqrt{k}\ \sigma(x)dW_\tau$. For more details, see **Example 5**).

However, in cases where a parametric model is preferred and a small search region is chosen, incorrect solutions may still occur. For instance, in this example you might also get the following wrong result (roughly speaking, with 5 `search_agents` and the considered searching region, you can expect to obtain the correct answer approximately 80% of the time)

```
Estimated parameters :
0.0405253      4.52763    0.0252577      83.2777     0.400786
- sum of log-likelihoods) : -89896.9211
```

But we have a criterion to determine the correct solution: the solution with the smallest objective value is the fittest.

Example 5. Now, let's proceed to fit a spline model. Type the following lines

```
S = load('MayData1D.mat');data = S.data;  %load the data
data = data(1:100000); %We only use the first third of the data
dt = 0.01;
L = min(data);R = max(data);
mu = 8;sigma = 8; % A spline model with 8 knots for mu and 8 knots for sigma
result5 = euler_reconstruction(data, dt, 'nKnots', [mu sigma], 'spline', 'CC', 'L',
...
L, 'R', R, 'lb', [zeros(1, mu) - 10, zeros(1, sigma)+eps], 'ub', zeros(1, mu + sigma)
+ 10, 'solver', 'fmincon', 'search_agents', 1);
```

and, you always get the following result

```
Estimated parameters :
0.4263    0.10858   -0.09268   0.0019837   0.054657   0.051057   -0.18033    -0.58468
0.37525   0.39834    0.39633  0.4086       0.40105    0.40098     0.39824     0.3948
- sum of log-likelihoods) : -180071.2916
```

and to see a plot type the following (see **Figure 3**, right panel)

```
r=1.01;K=10;g=2.75;a=1.6;s=0.4;  % true parameter values
par = [r K g a s];
mu = @(x,par)r.*x.*(1-x./K)-g.*x.^2./(x.^2+a.^2);sigma=@(x,par)s;  % true model
xplot=linspace(L,R,2000);
plot_results(result5,xplot,mu(xplot,par),sigma(xplot,par))
```

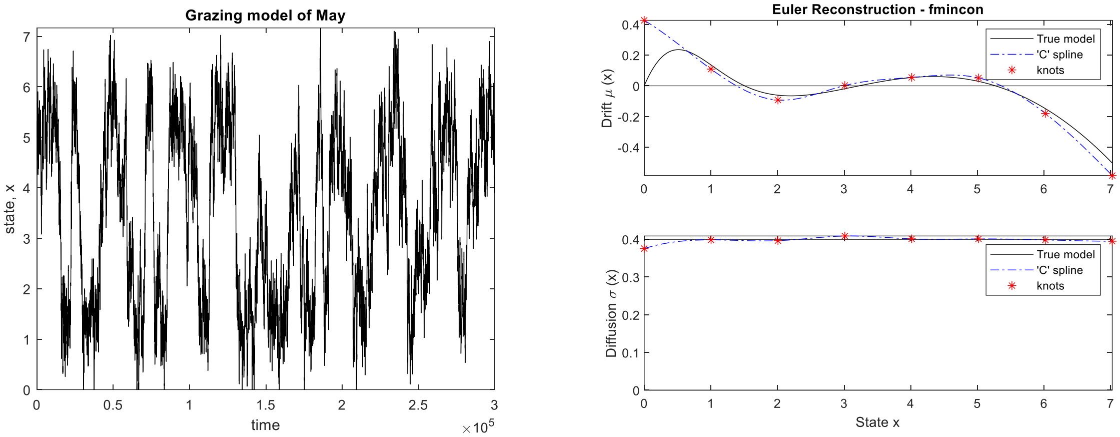


Figure 3. (Left panel) A dataset simulated from the grazing model of May. 3×10^5 data points with a time step of $dt = 0.01$ are simulated from the overgrazed model of May $dx = \left\{rx\left(1 - \frac{x}{K}\right) - \frac{\gamma x^2}{x^2 + a^2}\right\} dt + \sigma\, dW$ with parameters $r = 1.01, K = 10, \gamma = 2.75, a = 1.6$, $\sigma = 0.4$**.** (Right panel) A graphical illustration for a spline model with 8 knots (red stars) considered for both drift and diffusion functions (dot-dashed blue curves) together with the true model (solid black curves).

Some explanations for this spline model. First, we did not specify any lower and upper bound for the dataset since in this case there exist enough data across the state space. In such cases the solver chooses the default option [`min(data), max(data)`] for the state space. Second, we only chose one `search_agents`. However, note that this spline model has 16 parameters, yet a single search agent was sufficient to obtain the only solution. Third, this problem has only one solution. This highlights the simplicity of working with spline models, which are recommended. Splines are composed of simple polynomial building blocks and are flexible structures capable of adapting to complex functional forms with unknown nonlinearities. Consequently, spline modeling imposes less pressure on the optimization problem and often yields a unique solution. Forth, in spline modeling we do not use gradient. Calculating a gradient vector computationally requires $O(n)$ operations where n is the number of parameters. However, the computational time to estimate them numerically using MATLAB's finite-difference methods requires roughly the same operations. As the underlying MLE procedure for spline models is easier, we opt not to pass a gradient. Fifth, in general, selecting a proper model for parametric modeling can be challenging. Therefore, we emphasize the importance of the spline modeling approach. Even if there is a preference for a parametric model, we recommend starting with a spline model to gain insights into the functional form. Attempting to fit an improper parametric model to the data can result in issues such as longer execution times, decreased accuracy, and stagnation.

In **Figure 3**, the left tail of the May model did not appear in the reconstructed model. This is not related to stuff like numerical inaccuracies, estimation errors, etc. Indeed, this phenomenon occurs with any positive dataset generated by the May model. For an explanation about this refer to section12.

14.3 Reconstructing an ecological dataset

Example 6. Here, we apply a spline reconstruction to a univariate time series of phycocyanin concentrations in Lake Mendota (Carpenter et al. 2020). This dataset has a high resolution, with measurements taken at minute intervals. We focus on a period during summer thermal stratification in 2011, a period when Cyanobacterial blooms are common (see **Figure 4**, left panel). For further details on this dataset we refer you to the references (Arani et al. 2021, Magnuson et al. 2023). We do not standardize this real dataset since it is already the standardized level of phycocyanin concentrations (for further details on this read the appendix of (Arani et al. 2021)). Furthermore, this dataset does not satisfy the data requirements mentioned in section 6.3 since it is not Markov. However, a rarified sample of this dataset with every third data point (which is still high resolution) is Markov as ME time scale is 3 (see Table 1). We then apply Euler reconstruction to this sample. Type the following command lines

```
data = readmatrix('BGA_stdlevel_2011.csv');
data = data(:,3);
data = data(1:3:end); % From Table1, we see that this dataset is not Markov. But, a
rarified sample with every third data point is Markov
dt = 1;  % This is completely arbitrary.
L = -6.5;R = 6;
mu = 8;sigma = 8; % A spline model with 8 knots for mu and 8 knots for sigma
result6 = euler_reconstruction(data, dt, 'nKnots', [mu sigma], 'spline', 'CC', 'L',
...
L, 'R', R, 'lb', [zeros(1, mu) - 10, zeros(1, sigma)+eps], 'ub', zeros(1, mu + sigma)
+ 10, 'solver', 'fmincon', 'search_agents', 5);
```

which leads us to the following solution

```
Estimated parameters:
0.2961   0.098078 -0.091456 -0.048912  6.1932e-06 0.028938 -0.073625 -0.48324
0.76192  0.42613   0.48327   0.45969   0.35586    0.30587   0.3841    0.53274
- sum of log-likelihoods): 24958.2769
```

Finally, to get a plot type the following commands (see **Figure 4**, bottom panels)

```
xplot = linspace(L,R,2000);
plot_results(result6,xplot)
```

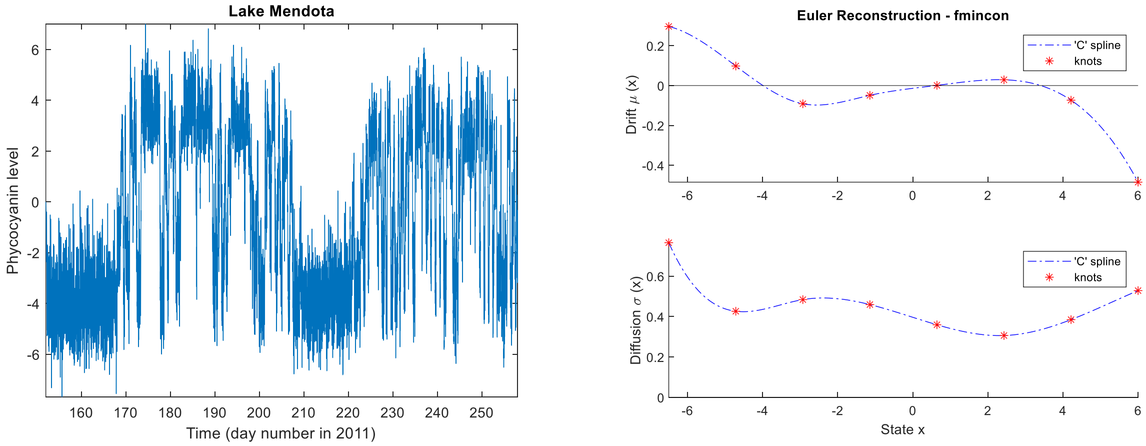


Figure 4. Application of spline modelling to a real dataset. The left panel illustrates a high-resolution cyanobacterial dataset measured at lake Mendota. While this dataset does not meet the data requirements outlined in section 6.3 (refer to **Table 1**), a rarified sample of this dataset, including every third data point, satisfies these requirements. The right panel illustrates the estimated drift and diffusion functions using spline modeling.

15. Univariate sparsely sampled data: Hermite reconstruction

It is not uncommon to encounter datasets with low-resolution especially in life science, presenting a challenge for reconstruction techniques like Euler reconstruction, which typically require at least a medium resolution for effective results. To address the issue of sparsely sampled data, our package implements a reconstruction technique based on a refinement approach by Bakshi, et.al (Bakshi and Ju 2005), building upon the work of Aït-Sahalia (Aït-Sahalia 2002) for univariate data. Aït-Sahalia's method involves constructing a sequence of converging closed-form Hermite expansions of 'transition density' $p(x(t+\Delta)|\ x(t))$ for which there is no closed form in almost all stochastic models. Therefore, we call it 'Hermite reconstruction'. Hermite reconstruction offers a higher level of accuracy but comes at a higher computational cost compared to the Euler reconstruction. We use Hermite reconstruction when the data resolution is low, resulting in an inaccurate reconstruction by the Euler approach. In such cases, we turn to Hermit reconstruction which has the capacity to enhance the accuracy of Euler reconstruction to some extent. In particular, we could improve poor outcomes of Euler reconstruction applied to low-resolution data when the model is linear. For nonlinear models the degree of improvement is usually less.

When implementing Hermite reconstruction, our package requires Euler parameter estimation as a starting solution. Here, the procedure has two phases in which in the first phase we use Euler reconstruction. In the second phase, the package follows Hermite reconstruction aims to enhance Euler estimation by exploring a 'small' region in the parameter space around the Euler estimation. Hermite reconstruction requires two key inputs: J and K which determine the number of terms one includes in the Hermite expansion of transition density (and hence the likelihood function). J represents the number of 'spatial' terms in the Hermite expansion of the likelihood function using Hermite polynomials, while K represents the number of 'temporal' terms in the Taylor expansion, in terms of sampling time dt, of Hermite coefficients. High J and/or K increase estimation accuracy at the cost of higher computation time. Typically, a small J suffices (for technical details, see the last paragraph on page 2 of Aït-Sahalia's paper (Aït-Sahalia 2002)), and in all case studies in the tutorial, we have used J=3. However, as data resolution decreases, a bigger K is necessary to enhance estimation accuracy. Based on our experience, for data with low-to-medium resolution, a value of K≤6 is generally sufficient while for low-resolution data values of 6<K≤12 may be necessary. However, we do not recommend using K>9 unless the model is simple (e.g., linear in parameters with a low number of parameters, such as the OU model. See **Example 7**). Using larger K values can lead to a complex optimization problem (i.e., the MLE) with numerous local minima. In practice, we often use J=3 and K=9 for low-resolution data and this value was applied in all the examples involving spline modeling presented in this paper.

15.1 When does Hermite reconstruction crash? Strategies and precautions

Hermite reconstruction works by constructing a sequence of closed-form expansions of transition density using Hermite polynomials. However, this expansion does not converge to s positive density, leading to an undefined objective value, in situations where data resolution is very low and initial parameters are rather far from the optimal parameters. To inspect this, the package attempts to find some starter parameter values in the vicinity of the Euler solution obtained in the first phase, which we call '*legitimate points*', where the objective function is defined. The function `legitimate_points` is responsible for this task. The challenge lies in finding the first legitimate point. Therefore, whether or not the package is able to tackle Hermite reconstruction boils, primarily, down to finding this first legitimate point (a feasibility problem). If the package can find it, then the problem is often tractable; otherwise, the problem is considered intractable based on the modeling strategy adopted (although the package might take a while to find the first legitimate solution, subsequent ones will be found faster). If the former occurs, the package uses the legitimate points first and applies a surrogate optimization for a very short time. Surrogate optimization is a technique used to optimize complex, computationally expensive, black-box, or undefined objective functions by replacing them with simpler surrogate models that approximate their behavior (Koziel and Leifsson 2013). Note that surrogate optimization is not among the optimization solvers you can use, rather it is an internal optimization which is used by the package. These surrogate models are typically easier to solve. Surrogate optimization then gives us a solution. If surrogate optimization finds a solution better than all legitimate solutions (as indicated by a message in the command window), then the package seeks a further improvement starting from the surrogate solution, using the solver chosen by the user which is either `fmincon` (recommended) or `gwo`. Otherwise, the package tries to make progress using the legitimate solutions as starters. However, if the package either fails to find legitimate solutions or takes a considerable time to do so, it suggests that the objective function is severely damaged (a message in the command window appears when a legitimate solution is found, providing the user with insight into the package's efficiency in finding legitimate solutions). In such cases, it is advisable to consider changing the modeling strategy by trying to fit a simpler model to data. *The best advice is to use quadratic spline modeling, opting for* `'QQ'` *flag (or, simply* `'Q'` *flag if additive modeling* *is preferred) instead of cubic spline modeling*. This significantly reduces the computational burden and increases the likelihood of the package in finding legitimate solutions. However, if you insist to use cubic spline modeling (though not recommended), it is advised to prioritize additive spline modeling over multiplicative modeling, despite potential limitations in efficiency. Additionally, opting for models that are linear in parameters is beneficial. Spline modeling becomes significant here, as splines are linear in terms of parameters while being nonlinear in terms of the state variable. Another valuable suggestion is to reduce the number of parameters in your model. This aids the package in finding an initial legitimate point in the parameter space more efficiently. A smaller number of parameters means that the package needs to search within a smaller space, thus increasing the chance of success. We illustrate these issues through several examples in this section, where Hermite reconstruction is applied to three datasets: one generated by a linear model, another by a nonlinear model, and a third from ice-core climate data.

15.2 Reconstructing a low-resolution dataset simulated from a linear model

Example 7. In this example, we apply Hermite reconstruction to the same dataset as in **Example 1** which was generated from the OU model $dx = \mu x\,dt + \sigma dW$ with parameters $\mu = -1$ and $\sigma = 1$. The dataset has a time step of `dt=0.01` and contains `T=10`^{6} data points. However, we select every 100th data point, resulting in an extremely low-resolution dataset with a time step of `dt=1` and `T=10`^{4} data points. The reason for this dataset to have an 'extremely' low-resolution is as follows: the relaxation time step for this dataset is `1` (see subsection 6.3 for more technical details on the data resolution). To see this, type the following commands

```
S = load('OUdata1D.mat');
data = S.data;
data = data(1:100:end);
R = RelaxationTime(data);R

1.0073
```

Where `R` should, in theory, be `1` if we had a longer dataset. This relaxation time signifies the lowest resolution in theory, where all reconstruction procedures fail if the data resolution is lower than this extreme value. Now, we perform Euler reconstruction (first phase). Type the following commands

```
S = load('OUdata1D.mat');
data = S.data;  %load the data
data = data(1:100:end); % Only every 100 data points are considered
dt = 1; % note that the mother dataset has the time step of dt=0.01 which is multiplied
%by 100 to match the time scale of this sample
mu = @(x,par)par(1).*x;sigma = @(x,par)par(2);
result10 = euler_reconstruction(data, dt, 'mu', mu, 'sigma', sigma, 'gradient_fun',
eulergrad(mu, sigma), 'lb', [-200 eps], 'ub', [200 200],'useparallel',true, 'solver',
'fmincon', 'search_agents', 5);
```

and you get

```
Estimated parameters :
-0.63555     0.65456
- sum of log-likelihoods): 9950.4726
```

This estimate deviates significantly from the true parameter values, primarily due to the fact that we rarified the original dataset to create this low-resolution dataset. This (Euler reconstruction) marks the completion of the first phase. Moving on to the second phase, we employ Hermite reconstruction, wherein the package explores in the vicinity of the Euler estimation to improve it. At this stage you have two choices to pick an optimization solver: `fmincon` or `gwo`. Opting for `fmincon` (recommended) involves an initial step of running the function `legitimate_points` to find at least `N = search_agents` legitimate starting points before proceeding to solve the underlying optimization problem (MLE). Subsequently, you can execute the main function `hermite_reconstruction` to estimate the optimal parameter values. This sequential approach is essential because `fmincon` cannot initiate optimization with an infeasible solution, although it can fortunately cope with infeasible solutions, to some extent, all the way to the optimal parameter values. However, in cases of very low-resolution data and nonlinear models, there's a higher risk of the fmincon solver crashing, whereas the gwo solver, although resilient to crashes, may be considerably slower in such scenarios. Therefore, our recommendation remains to first attempt fmincon. Nonetheless, if you prefer to utilize the gwo solver, you can directly proceed with the hermite_reconstruction function and skip executing legitimate_points since gwo internally calls it. Another consideration is the selection of two parameters J (J≥ 3) and K (K≥ 1), which are required for the implementation of Hermite reconstruction. ***Typically, J=3 is sufficient***. However, for enhanced parameter estimation, a larger value of K is necessary. ***As a rule of thumb, for datasets with medium resolution (see subsection 6.3 on data resolution) K≤ 6 should suffice, while for extremely low-resolution data, values of 6 <K≤ 12 may be warranted***. Here, we consider K=9. Now, let's proceed with fmincon. Type the following command.

```
legpoints = legitimate_points(data, dt, 'prev', result10, 'prev_range', 0.5, 'j', 3,
'k', 9);
```

Now, type the following command

```
result_her10 = hermite_reconstruction(data, dt, 'prev', legpoints,'solver', 'fmincon');
```

and you get the following great result

```
Estimated parameters:
-0.99686 0.9949
- sum of log-likelihoods: 9950.8106
```

Table 2 summarizes the results obtained for various values of K (for a fixed J=3) within the range 1 <K≤ 12.

K	Estimated μ, par(1)	Estimated σ, par(2)	Objective value (Negative sum of log-likelihoods)
Euler	-0.63555	0.65456	—
1	-0.020199	0.8149	11774.4678
2	-0.2273	0.83345	11000.0335
3	-0.30889	0.87803	10728.8377
4	-0.48813	0.87103	10326.1415
5	-0.64935	0.90212	10111.3767
6	-0.71338	0.93515	10060.9384
7	-0.85559	0.95665	9978.7035
8	-0.93911	0.97968	9960.3823
9	**-0.99686**	**0.9949**	**9950.8106**
10	-1.0059	0.9977	9950.8619
11	-1.0099	0.99885	9950.402
12	-1.0092	0.99865	9950.4912

Table 2. A summary of the parameter estimations as we vary K within the range 1 <K≤ 12 for a fixed J=3.

Note that in general when data resolution is low, we cannot compare the objective values for the Euler solution with other objective values in **Table 2**.

Example 8. Let's now try to fit a spline model to the same dataset in the **Example 7**. For the first phase type the following commands

```
S = load('OUdata1D.mat');data = S.data;  %load the data
data = data(1:100:end); % Only every 100 data points are considered
L = -2.5;R = 2.5;%Since we have a  spline  model it is better to shrink the state space
```

```
data(data<L | data>R) = nan;  %This is VERY important: In spline modeling if you
consider a smaller range for your data then you must assign ‘nan’ to those few data
points falling outside this range.
dt = 1;
mu = 7; sigma = 7; %In spline modeling mu and sigma are numbers
result11 = euler_reconstruction(data, dt, 'nKnots', [mu sigma], 'spline', 'QQ', 'L',
L, 'R', R, ...,
'lb', [zeros(1, mu) - 10, zeros(1, sigma)+eps], 'ub', zeros(1, mu + sigma) + 10,
'solver', 'fmincon', 'search_agents', 5);
```

And you always get the following solution

```
Estimated parameters:
1.7415    1.0522    0.52421    -0.0032876   -0.52616   -1.1204     -1.6048
0.38686   0.65803   0.65051     0.66209      0.63977    0.63572     0.48534
- sum of log-likelihoods): 9920.7613
```

It is important to note that for this example, we have adjusted the range of the data due to the implementation of spline modeling. Typically, the range of state space is, by default, `[min(data) max(data)] = [-2.6289 2.9383]`. However, in this example, we have narrowed it down to `[-2.5 2.5]`. This decision was made because the dataset contains only 2 data points larger than 2 and 2 data points smaller than -2 out of a total of 10,000 data points. This situation often arises with small datasets that have few data points near the data borders. Failing to address this issue can negatively impact the estimation process in terms of both accuracy and speed, particularly during the second phase when Hermite reconstruction is implemented. Additionally, note that we've assigned `nan` values to those few data points that fall outside our considered range. Here, we have considered the spline flag `'QQ'` and considered `mu = 7` and `sigma = 7`. We elaborate on these choices later. To implement the second phase, first type the following command

```
legpoints = legitimate_points(data, dt, 'prev', result11, 'prev_range', 0.5, 'j', 3,
'k', 9);
```

which provides us with $N \geq$ search_agents = 5 legitimate solutions. Finally, type the following command

```
result_her11 = hermite_reconstruction(data, dt, 'prev', legpoints, 'solver', 'fmincon',
'search_agents', 5);
```

and you get

```
Estimated parameters:
2.2275     1.6342     0.86879     0.020833    -0.8565     -1.6008     -2.1243
0.92026    1.0035     1.0253      1.0373       1.0147      0.96308     0.86743
- sum of log-likelihoods: 9913.6229
```

Unlike Euler spline reconstruction, Hermite spline reconstruction may yield slightly different results. This variability is not necessarily due to do a multiplicity of local minima, but rather to the quality of the legitimate solutions obtained, which can impact the capacity of the package to solve non-smooth optimization problems. However, other solutions closely approximate this one. Several points are noteworthy here. First, the lower objective value in this example (`9913.6229`) compared to **Example** `7` (`9950.8106`) indicates that the model in this example is a better fit. This outcome was expected as the model in this example comprises `mu + sigma = 14` parameters, whereas the model in **Example** `7` had only `2` parameters. Second, here we have chosen the spline flag `'QQ'` not the typical flag `'CC'` we used before. If the user only wishes to apply Euler reconstruction, then this choice does not matter a lot so, one can safely use the flag `'CC'`. If, however, the final goal is to apply a Hermite reconstruction to data then it is recommended to use the spline flag `'QQ'`. This greatly speeds up the calculations, increases the chance of finding legitimate solutions and, further increases the chance of improving the legitimate solutions later. Third, in this example, we fit a multiplicative spline model to data using `mu = 7` knots for the drift function and `sigma = 7` knots for the diffusion function. For Hermite reconstruction, we need to be economical with respect to the number of parameters. In this example, attempting `mu = 8` and `sigma = 8` also works but the computational time increases and the package has difficulty in making a progress (while it took around 20 seconds to find 5 legitimate solutions here, it would take a few minutes for the case `mu = 8` and `sigma = 8`). Furthermore, if the spline flag `'CC'` would be used, instead, it would be very hard to find legitimate solutions beyond the cases where `mu≥5` and `sigma≥5` in a reasonable time. *A general advice, therefore, is to use the spline flag `'QQ'` and be economical on the number of parameters*. Otherwise: 1) it takes a long time for the package to find legitimate solutions, and 2) the package fails to improve upon the Euler reconstruction later. Due to the 'curse of dimensionality', as the parameter space expands, finding legitimate solutions becomes increasingly challenging. This is so because the illegitimate points densely populate the objective function, making it difficult for the `fmincon` solver to progress in the optimization process. If the density of illegitimate points is not too high (roughly <25%), the `fmincon` solver can manage, but otherwise, it becomes stuck. Unfortunately, the `gwo` solver also cannot help in a short time in such cases. If you wish to consider a bigger knot sequence then a good strategy is to consider an additive model (see the next example). To get a plot, type the following commands

```
mu = @(x,par)par(1).*x;sigma = @(x,par)par(2)+0.*x; %this is true model
par = zeros(2,1);par(1) = -1;par(2) = 1; %true model parameters
xplot = linspace(L,R,2000); % a dense mesh across the considered range
plot_results(result11,xplot,mu(xplot,par),sigma(xplot,par)); % Euler & true models
plot_results(result_her11,xplot,mu(xplot,par),sigma(xplot,par));%Hermite & true models

```

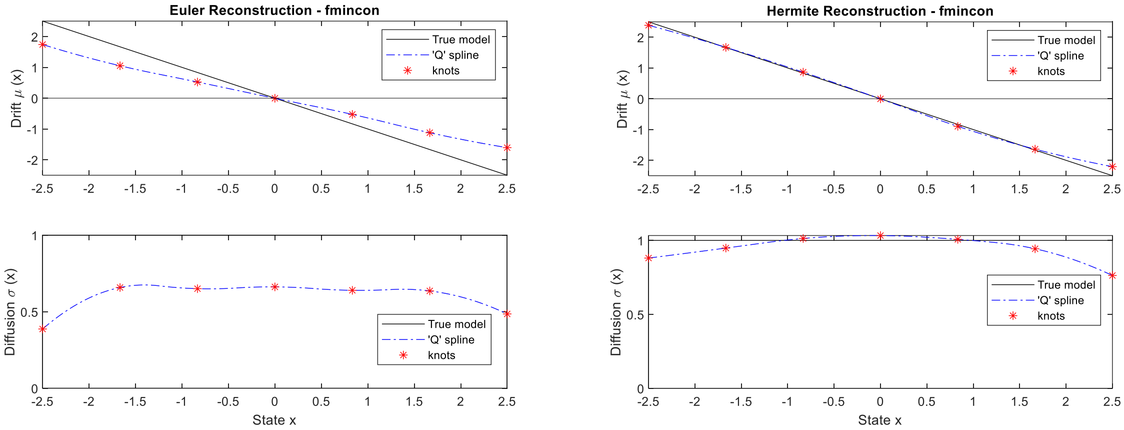


Figure 5. Illustration of Euler and Hermite multiplicative reconstructions applied to a low-resolution simulated dataset generated by a linear model. The left panel depicts the true model alongside the Euler reconstructed model, while the right panel depicts the true model alongside the Hermite reconstructed model. The data are simulated from the OU model $dx = \mu x\,dt + \sigma dW$ with parameters $\mu = -1$ and $\sigma = 1$. The original dataset has the time step of `dt=0.01` and contains `T=10`^{6} data points. However, we select every 100th data point, resulting in an extremely low-resolution dataset with a time step of `dt=1` and `T=10`^{4} data points.

Example 9. Type the following command lines (we omit the explanatory details)

```
S = load('OUdata1D.mat');data = S.data;  %load the data
data = data(1:100:end); % Only every 100 data points are considered
L = -2.5;R = 2.5; %Since we have a  spline  model it is better to shrink the state
space
data(data<L | data>R) = nan;  %This is VERY important: In spline modeling if you
consider a smaller range for your data then you must assign 'nan' to those few data
points falling outside this range.
dt = 1;
mu = 8; sigma = 1; %In spline modeling mu and sigma are numbers
result12 = euler_reconstruction(data, dt, 'nKnots', [mu sigma], 'spline', 'QQ', 'L',
L, 'R', R, ...,
'lb', [zeros(1, mu) - 10, zeros(1, sigma)+eps], 'ub', zeros(1, mu + sigma) + 10,
'solver', 'fmincon', 'search_agents', 5);

legpoints = legitimate_points(data, dt, 'prev', result12, 'prev_range', 0.5, 'j', 3,
'k', 9);
result_her12 = hermite_reconstruction(data, dt, 'prev', legpoints, 'solver',
'fmincon', 'search_agents', 5);

mu=@(x,par)par(1).*x;sigma=@(x,par)par(2)+0.*x; %this is true model
par=zeros(2,1);par(1)=-1;par(2)=1; %true model parameters
xplot=linspace(L,R,2000); % a dense mesh across the considered range
plot_results(result12,xplot,mu(xplot,par),sigma(xplot,par)); % Euler & true models
plot_results(result_her12,xplot,mu(xplot,par),sigma(xplot,par));%Hermite & true
models
```

Figure 8 illustrates the outcomes of Euler, Hermite and true models. In this additive model, the Hermite objective value (9918.9198) is higher than that in the multiplicative model in **Example 8** (9913.6229) which means that the multiplicative model is a better fit. However, it was much faster for the package to handle this example since this model has less parameters.

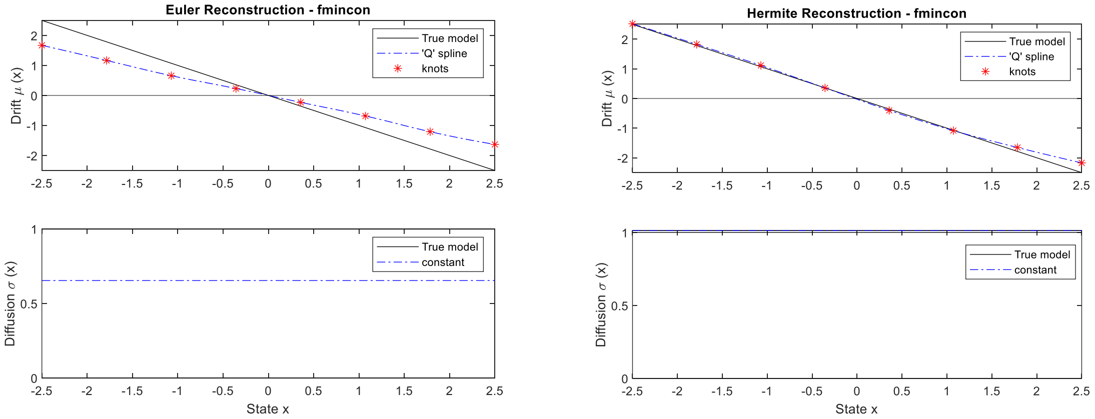


Figure 5. Illustration of Euler and Hermite additive reconstructions applied to a low-resolution simulated dataset generated by a linear model. The left panel depicts the true model alongside the Euler reconstructed model, while the right panel depicts the true model alongside the Hermite reconstructed model. The data are simulated from the OU model $dx = \mu x\, dt + \sigma dW$ with parameters $\mu = -1$ and $\sigma = 1$. The original dataset has the time step of `dt=0.01` and contains T=10^6 data points. However, we select every 100th data point, resulting in an extremely low-resolution dataset with a time step of `dt=1` and T=10^4 data points.

15.3 Reconstructing a low-resolution dataset simulated from a nonlinear model

Here, we apply Hermite reconstruction to the following nonlinear model which is a stochastic version of the overgrazed model of May(May 1977)

$$dx = \left\{ rx\left(1-\frac{x}{K}\right) - \frac{\gamma x^2}{x^2+a^2}\right\} dt + \sigma\, dW,$$

where the model parameters are $r = 1.01, K = 10, \gamma = 2.75, a = 1.6,\ \sigma = 0.4$. We have simulated a dataset containing $3 * 10^5$ data points with time step $dt = 0.01$. We consider estimating the parameters of this model by rarifying this data set by considering every 300th data points to get a sparse sample with time step $dt = 3$ and only 1000 data points. Let’s first check the resolution of this dataset via its relaxation time. Type the following commands

```
S = load('MayData1D.mat');data = S.data;  %load the data
data=data(1:300:end); %We consider every 300-th data point
RelaxationTime(data)
```

and you get

```
12.7959 (unit of data)
```

which puts this rarified dataset in the category of low-resolution. We recall that a dataset with relaxation time in the interval [1 50] is considered low-resolution (see subsection `6.3` for more details).

Example 10. Here, we try to fit a parametric model to this dataset. If we try a model with drift term `mu = @(x,par)par(1).*x.*(1-x./par(2))-par(3).*x.^2./(x.^2+par(4).^2)` then Hermite reconstruction cannot improve the Euler reconstruction. The core of difficulty is that this model is nonlinear in terms of two parameters: `par(2)` and `par(4)`. We, therefore, try to fit a nonlinear model which is linear in terms of parameters. Type the following commands

```
S = load('MayData1D.mat');data = S.data;  %load the data
data=data(1:300:end); %We consider every 300-th data point
dt = 3;  % Since in the mother dataset dt=0.01 and here we considered every 300-th
data points the actual time step is 3
m = mean(data);s = std(data);
mu = @(x,par)par(1).*((x-m)./s).^3+par(2).*((x-m)./s).^2+par(3).*(x-m)./s+par(4);
%this is a standardized drift model
sigma = @(x,par)par(5);
result13 = euler_reconstruction(data, dt, 'mu', mu, 'sigma', sigma, 'gradient_fun',
eulergrad(mu, sigma), ...
    'lb', [-5.*ones(1,4) 0], 'ub', [5.*ones(1,4)
5],'useparallel',true,'search_agents', 5);  % Since the model lb for drift parameters
is chosen to be symmetrical about 0, i.e., the interval [-5 5]
```

```
S = load('MayData1D.mat');data = S.data;  %load the data
data = data(1:300:end); %We consider every 300-th data point
data = (data-mean(data))./std(data);
dt = 3;  % Since in the mother dataset dt=0.01 and here we considered every 300-th
data points the actual time step is 3
mu = @(x,par)par(1).*x.^3+par(2).*x.^2+par(3).*x+par(4);sigma=@(x,par)par(5);
result13 = euler_reconstruction(data, dt, 'mu', mu, 'sigma', sigma, 'gradient_fun',
eulergrad(mu, sigma),'lb', [-5.*ones(1,4) 0], 'ub', [5.*ones(1,4)
5],'useparallel',true,'search_agents', 5); %Since data are standardized the vectors of lower and
upper bounds for the drift part are better to be symmetrical about 0, which is [-5 5] here. For the
diffusion parameters it should be [0 5]
```

we then obtain

```
Estimated parameters:
-0.038801   -0.0045207    0.041769   0.0017958     0.20353
- sum of log-likelihoods): 375.9106
```

Here, we provide explanations. First, it is important to note that the original dataset has the time step of $dt = 0.01$. However, since the rarified dataset considers every 300th data points from the original dataset, its time step is $dt = 3$. As mentioned previously, the choice of time step during the reconstruction process acts as a scale parameter and is completely arbitrary. Nonetheless, for the sake of validating our approach, we opt to use the actual time steps. Second, to streamline the optimization process, we standardized the data. Consequently, the corresponding parameters can vary around `0`, and their magnitudes are not significantly different from `0`. Therefore, we used a symmetrical lower and upper bound for each drift parameters, i.e., the interval `[-5 5]`. For the noise parameter, we need to consider the interval `[0 5]` to ensure the positivity of the diffusion function. Finally, in order to recover a model that accurately represents the original data, we must back-transform the parameters to their original scales. This involves subtracting the state variable by the mean of the data, dividing by the standard deviation of the data, and then multiplying the entire system by the standard deviation of the data. Consequently, the following drift and diffusion functions model the original data

```
mu = @(x,par)s.*par(1).*((x-m)./s).^3+s.*par(2).*((x-m)./s).^2+s.*par(3).*(x-
m)./s+s.*par(4);sigma=@(x,par)s.*par(5)
```

where `m = mean(data)` and `s = std(data)`. Next, we get 5 legitimate points by the following command

```
legpoints = legitimate_points(data, dt, 'prev', result13, 'prev_range', 0.5, 'j', 3,
'k', 9);
```

and, finally we go for Hermite reconstruction as bellow

```
result_her13 = hermite_reconstruction(data, dt, 'prev', legpoints, 'solver',
'fmincon', 'search_agents', 5);
```

to obtain

```
Estimated parameters:
-0.044756 -0.0042904   0.0464   0.0022962     0.22073
- sum of log-likelihoods: 362.5456
```

The lower objective value for the Hermite reconstruction (`362.5456`) compared to the Euler reconstruction (`375.9106`) indicates an improvement in the parameter estimation. To generate plots for both the Euler and Hermite outcomes, type the following commands (see **Figure 6**)

```
result = result13; %plot for Euler outcomes
S = load('MayData1D.mat');data = S.data;
m = mean(data);s = std(data);
result.s = s;result.m = m;result.par_est = result.estimated_par;
result.mufun = @(x,par_est)s.*par_est(1).*((x-m)./s-par_est(2)).*((x-m)./s-
par_est(3)).*((x-m)./s-par_est(4)); %back-transformed drift (subtract state by data mean
and divide by data standard deviation. Finally multiply the whole by data standard deviation)
result.sigmafun = @(x,par_est)s.*par_est(5)+0.*x; %back-transformed diffusion
mu = @(x,par)par(1).*x.*(1-x./par(2))-par(3).*x.^2./(par(4).^2+x.^2); %true drift
sigma = @(x,par)par(5)+0.*x; %true diffusion
par = zeros(5,1);par(1) = 1.01;par(2) = 10;par(3) = 2.75;par(4) = 1.6;par(5) = 0.4;
%true model parameters
xplot=linspace(0,6,2000);
plot_results(result,xplot,mu(xplot,par),sigma(xplot,par));

result = result_her13; %plot for Hermite outcomes
S = load('MayData1D.mat');data = S.data;
m = mean(data);s = std(data);
result.s = s;result.m = m;result.par_est = result.estimated_par;
result.mufun = @(x,par_est)s.*par_est(1).*((x-m)./s-par_est(2)).*((x-m)./s-
par_est(3)).*((x-m)./s-par_est(4));
result.sigmafun = @(x,par_est)s.*par_est(5)+0.*x;
mu = @(x,par)par(1).*x.*(1-x./par(2))-par(3).*x.^2./(par(4).^2+x.^2);
sigma = @(x,par)par(5)+0.*x;
par = zeros(5,1);par(1) = 1.01;par(2) = 10;par(3) = 2.75;par(4) = 1.6;par(5) = 0.4;
xplot=linspace(0,6,2000);
plot_results(result,xplot,mu(xplot,par),sigma(xplot,par));
```

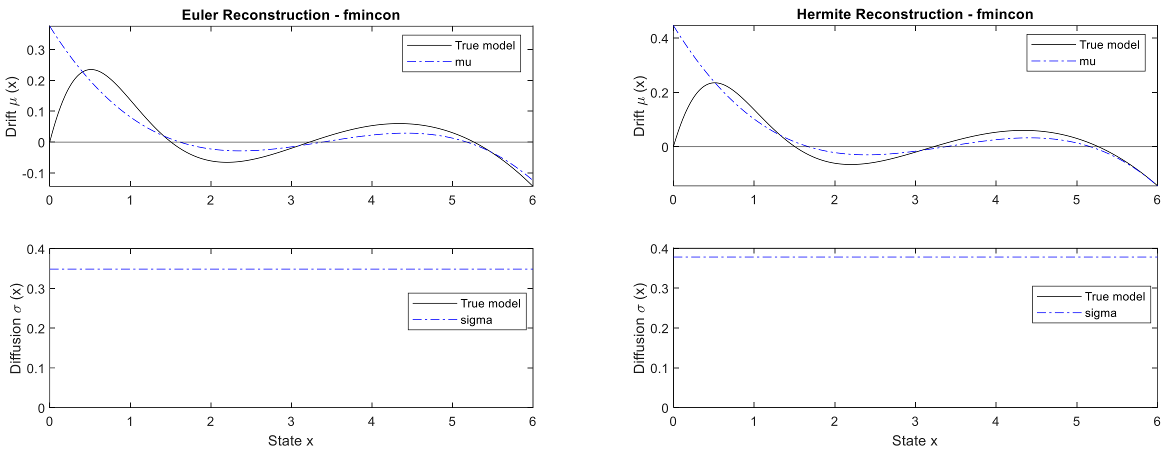


Figure 6. Illustration of Euler and Hermite reconstructions applied to a low-resolution simulated dataset generated by a nonlinear model. The left panel depicts the true model alongside the Euler reconstructed model, while the right panel depicts the true model alongside the Hermite reconstructed model. The data are simulated from the grazing model of May $dx = \left\{rx\left(1-\frac{x}{K}\right)-\frac{\gamma x^2}{x^2+a^2}\right\}dt+\sigma\, dW$ with parameters $r = 1.01, K = 10, \gamma = 2.75, a = 1.6,\ \sigma = 0.4$. The original dataset has the time step of `dt=0.01` and contains `T=3*10`^{3} data points. However, we select every 300th data point, resulting in a low-resolution dataset with a time step of `dt=3` and `T=10`^{3} data points.

As is evident from **Figure 6**, there is not a substantial improvement to the Euler outcomes after applying Hermite reconstruction. The biggest improvement is observed for the diffusion function, while the improvement in the drift function is slight. We improve these outcomes using spline reconstruction in the next example.

Example 11. Type the following command lines (we omit the details. See **Example 9** for more details)

```
S = load('MayData1D.mat');data = S.data;
data = data(1:300:end);
L = 0;R = max(data); %we consider the entire range of data
data(data<L | data>R) = nan;
dt = 3;
mu = 8; sigma = 8;
result14 = euler_reconstruction(data, dt, 'nKnots', [mu sigma], 'spline', 'QQ', 'L',
L, 'R', R, ...,
'lb', [zeros(1, mu) - 10, zeros(1, sigma)+eps], 'ub', zeros(1, mu + sigma) + 10,
'solver', 'fmincon', 'search_agents', 5);
legpoints = legitimate_points(data, dt, 'prev', result14, 'prev_range', 0.5, 'j', 3,
'k', 9);
result_her14 = hermite_reconstruction(data, dt, 'prev', legpoints, 'solver',...,
'fmincon');
```

these leads us to the following Euler parameter estimation

```
Estimated parameters:
0.31967   0.10093    -0.03319   -0.0049032   0.0088617   0.033111   -0.089981   -0.27115
0.19942   0.27834     0.36011    0.42682     0.35892     0.35962     0.33349     0.18335
- sum of log-likelihoods): 899.8273
```

and the following Hermite parameter estimation

```
Estimated parameters:
0.38364    0.10187   -0.032008   -0.029461    0.036058    0.025406    -0.09752    -0.35229
0.54784    0.38213    0.37651     0.40084     0.34381     0.39658      0.40582     0.29627
```

```
- sum of log-likelihoods: 889.6237
```

To plot the results, type the following command lines (see **Figure 10**)

```
mu = @(x,par)par(1).*x.*(1-x./par(2))-
par(3).*x.^2./(x.^2+par(4).^2);sigma=@(x,par)par(5)+0.*x;  % true model
par = zeros(5,1);par(1) = 1.01;par(2)= 10;par(3) = 2.75;par(4) = 1.6;par(5) = 0.4;
%true model parameters
xplot = linspace(L,R,2000); % a dense mesh across the considered range
plot_results(result14,xplot,mu(xplot,par),sigma(xplot,par)); % Euler & true models
plot_results(result_her14,xplot,mu(xplot,par),sigma(xplot,par));%Hermite & true models
```

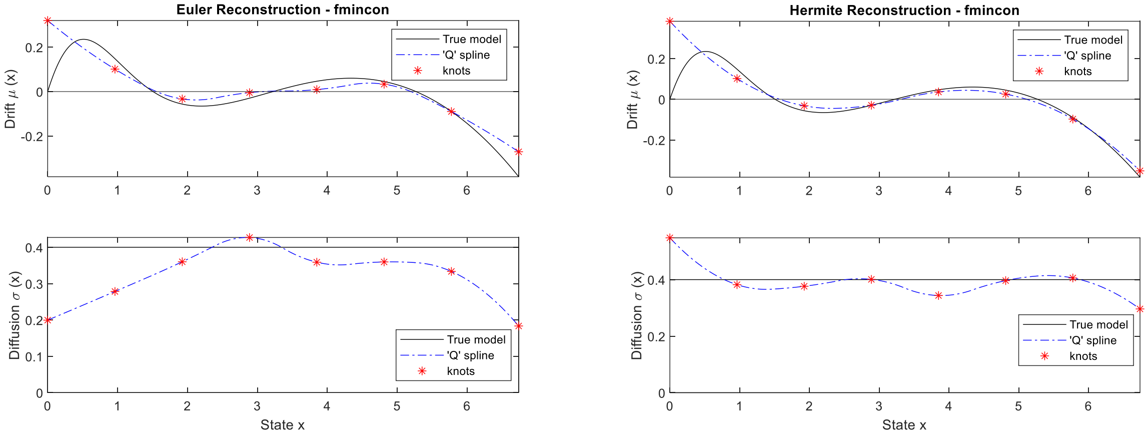


Figure 7. Illustration of Euler and Hermite spline reconstructions applied to a low-resolution simulated dataset generated by a nonlinear model. The left panel depicts the true model alongside the Euler reconstructed spline model, while the right panel depicts the true model alongside the Hermite reconstructed spline model. The data are simulated from the grazing model of May $dx = \left\{rx\left(1-\frac{x}{K}\right) - \frac{\gamma x^2}{x^2+a^2}\right\}dt + \sigma\, dW$ with parameters $r = 1.01, K = 10, \gamma = 2.75, a = 1.6$, $\sigma = 0.4$. The original dataset has the time step of `dt=0.01` and contains $\mathtt{T=3*10^3}$ data points. However, we select every 300th data point, resulting in a low-resolution dataset with a time step of `dt=3` and $\mathtt{T=10^3}$ data points.

You can improve the Hermite reconstruction slightly further by opting for larger `K` values, like `K = 10,11,12`, albeit with a slightly increased computational time.

15.4 Reconstructing a low-resolution ice-core dataset

Example 12. In this case study, we reconstruct a δ^{18}O record from the North Greenland Ice Core Project (NGRIP) (2004), which serves as a proxy for the temperature of the northern hemisphere. This record spans the last 120 thousand years, encompassing the last glaciation and has a resolution of 20 years. It is important to note that this resolution is different from the concept of resolution discussed in this tutorial (refer to subsection 6.3 for details). Due to the non-stationary nature of the dataset, our analysis is restricted to the time period from 70 to 20 thousand years before the present time (see **Figure 8**). For more details on this, refer to (Kwasniok and Lohmann 2009). Throughout the last glaciation, the climate of the northern hemisphere experienced alternating colder (stadial) and warmer (interstadial) states, due to a phenomenon called Dansgaard–Oeschger (DO) events (Dansgaard et al. 1993). Within the considered time window, the majority of DO events, DO2 to DO 18 out of 25 DO events, occurred (2004). The actual resolution of this dataset is 50 years, which we demonstrate to be low-resolution, comprising a total of 1001 data points. To assess the resolution of this dataset, we determine its relaxation time as below

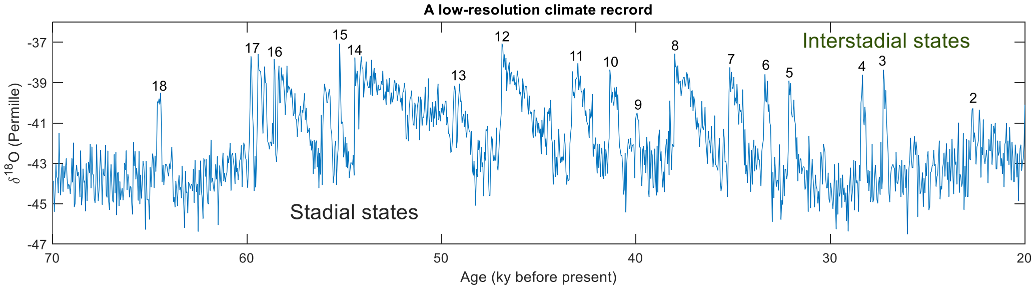


Figure 8. A δ^{18}O climate record, with a resolution of 20 years, extending from 70 to 20 thousand years before the present time from NGRIP. This is used as a proxy for the temperature of the northern hemisphere which shows that the northern hemisphere climate alternated between cold stadial and warmer interstadial alternative climate states. In this time period majority of Dansgaard-Oescher events, DO2 to DO18, occurred (see the numbers).

Although this dataset is not Markov, a sample containing every other point exhibit Markov property approximately (details in subsection 6.2 or **Table 1**). Additionally, due to its low resolution (discussed in subsection 6.3 or **Table 1**), Hermite reconstruction is necessary. To reconstruct this dataset, use the following commands (for details take a look at previous examples).

```
data = readmatrix('NGRIP20.csv');
data = data(2649:5081);
data = data(1:2:end);
L = -45.5;R = -38.2;
data(data<L | data>R) = nan;
dt = 1;
mu = 7; sigma = 7;
result15 = euler_reconstruction(data, dt, 'nKnots', [mu sigma], 'spline', 'QQ', 'L',
L, 'R', R, ...,
'lb', [zeros(1, mu) - 10, zeros(1, sigma)+eps], 'ub', zeros(1, mu + sigma) + 10,
'solver', 'fmincon', 'search_agents', 20); % we used 20 search agents
legpoints15 = legitimate_points(data, dt, 'prev', result15, 'prev_range', 0.5, 'j',
3, 'k', 9);
result_her15 = hermite_reconstruction(data, dt, 'prev', legpoints15, 'solver',
'fmincon');
```

you get the following Euler results

```
Estimated parameters :
2.5801    0.97206   0.0523    -0.50281    -0.18251    -0.23949    -1.1289
0.52863   1.079     1.0395    1.2871      1.0323      0.82494     0.78785
- sum of log-likelihoods) : 1721.4668

```

and the following Hermite results

```
Estimated parameters :
2.2923    1.1239   -0.017088    -0.45515    -0.093535    -0.29108    -1.3325
1.626     1.7236    1.5222       1.212       0.98925      0.92439     0.90594
- sum of log-likelihoods : 1651.6858
```

To depict the results type the following command lines (see **Figure**)

```
xplot = linspace(L,R,2000);
plot_results(result15,xplot);
plot_results(result_her15,xplot);
```

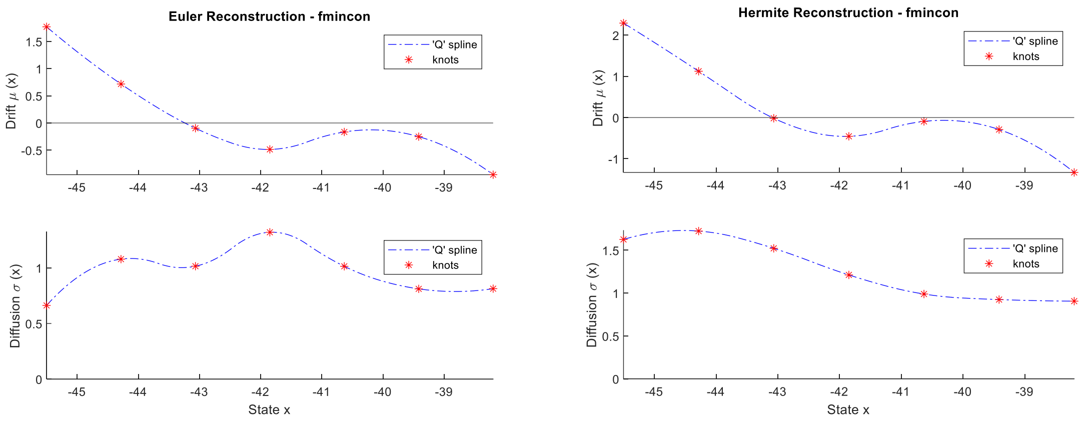


Figure 9. Illustration of Euler and Hermite spline reconstructions applied to a low-resolution ice-core climate dataset. The left panel depicts the Euler reconstructed spline model, while the right panel depicts the Hermite reconstructed spline model. The data are illustrated in **Figure 8**.

Some explanations are needed here. First, as mentioned, the entire dataset is not stationary, which is a data requirement in subsection 7.1. However, we have analyzed a portion of data that is stationary. If the analysis of the entire dataset is the goal, then the analysis performed on this portion should be repeated using a moving window approach. In this approach, one analyzes a short segment of data over a short window of time (which is often stationary), then shifts this time window slightly to the right and analyzes the second window, then shifts the second window slightly to the right and analyzes the third segment, so on. Eventually, you should get time-varying drift $\mu(x,t)$ and diffusion $\sigma(x,t)$ functions, which are calculated by interpolating the outcomes of all the segments. This is a simple scheme, and in practice, the size of segments does not need to be equal. Second, this dataset is not Markov and this is another data requirement we elaborated on in subsection 7.2. However, every other data point is nearly Markov, i.e., the ME time scale is 2 (see **Table 1**). Third, similar to **Example 8** we considered `mu = 7, sigma = 7`. Since, this is a very small dataset with 1217 data points if you consider 8 knots it also works but it takes a bit more time for the package to find the legitimate points. But, we believe 7 knots should be enough.

15.5 The concept of effective potential

Does the climate dataset in the previous subsection have ‘alternative stable states’? To address this question, we often attempt to find the roots of the drift function, i.e., solve for $\mu(x) = 0$. Following this approach, we indentify a single equilibrium near -43, as shown in **Figure 8**, bottom panel. However, this approach is incorrect. Such an approach is suitable for deterministic systems and stochastic systems with additive noise. Nevertheless, to accurately answer this question, it is necessary to calculate a quantity known as the ‘effective potential’ (refer to (Arani 2019, MS Arani et al. 2024) for further details)

$$U_{\text{eff}}(x) = -2\left(\int^{x} \frac{\mu(u)}{\sigma^2(u)}\,du + \log\sigma(x)\right), \tag{3}$$

where the integral in (3) is called ‘indefinite’ since it does not have a lower integration bound. To find the equilibria of a stochastic system we need information not only from the drift function $\mu(x)$ but also from the diffusion function $\sigma(x)$. Both pieces of information are incorporated into the effective potential $U_{\text{eff}}(x)$ in (3). Therefore, to determine the equilibria, we need to identify the minima and maxima of $U_{\text{eff}}(x)$, which correspond to stable and unstable equilibria, respectively. We will not delve into the details here, as the package can perform these calculations. To plot the estimated drift, diffusion, and effective potential functions for the Hermite reconstruction, type the following commands

```
xplot = linspace(L,R,2000);
plot_results(result_her15,xplot,'eff_potential');
```

and you get the following plot

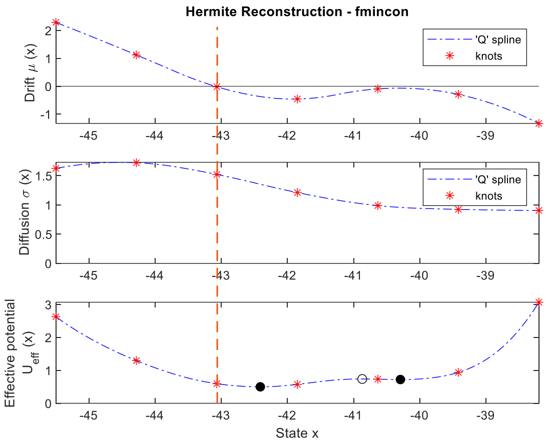


Figure 10. Illustration of the concept of effective and its significance in identifying alternative stable states in a stochastic system. The top and middle panels are as in **Example 12**, right panel. In the bottom panel the effective potential is depicted, with its minima and maxima corresponding to the stable (solid black dots) and unstable (open circle) equilibria. Notably, the dashed vertical dashed orange line intersects the only root of the drift function (i.e., where $\mu(x) = 0$) yet it does not coincide with any of the minima of the effective potential. This discrepancy suggests that relying solely on the drift function to calculate the equilibria is incorrect.

In **Figure 10**, bottom panel the vertical dashed orange line intersects the only root of the drift function (i.e., where $\mu(x) = 0$), but it does not coincide with any of the alternative stable states (i.e., the solid dots in **Figure 10**, bottom panel). This highlights that the only way to determine the equilibria of a stochastic system is to identify the minima and maxima of the effective potential.

16. Handling big datasets

When working with datasets containing millions of data points, the computational burden can be significant, leading us to consider using only a portion of the dataset. However, selecting the appropriate portion is crucial, as opting for the first 10%, last 10%, or middle portion can notably influence the final results, potentially introducing bias into the estimated parameters. Since diffusion models are Markovian, we can employ mini-batch optimization, where we sample a fraction of '*data pairs*' and solve the underlying optimization problem based on that fraction alone. Here, a 'data pair' refers to any consecutive pair (x_t, x_{t+1}) across the data. By randomly selecting a sample comprising just 10% of all data pairs, we can conduct the analysis on this subset. This fraction is well-mixed across the entire dataset and provides a representative sample. To ensure an even more random selection compared to simple random sampling, we recommend and implement a 'stratified' random sampling of data pairs. This method offers an excellent representation of the entire dataset. After a random sample of data is obtained, we can follow either of Euler or Hermite reconstruction as explained in previous sections.

Example 13. Consider the first dataset in **Example 1** which is simulated from the OU model. The length of this dataset is 10^6. Imagine that we just wish to perform parametric reconstruction using 1% of this dataset. Type the following commands

```
S = load('OUdata1D.mat');data = S.data;
dt = 0.01;  % the time step remains unchanged (this should not be confused with data
rarification)
mu = @(x,par)par(1).*x;sigma = @(x,par)par(2);
result16 = euler_reconstruction(data, dt, 'mu', mu, 'sigma', sigma, 'gradient_fun',
eulergrad(mu, sigma), ...
    'reconst_fraction', [10 0.01], 'lb', [-200 eps], 'ub', [200
200],'useparallel',true, 'solver', 'fmincon', 'search_agents', 5);
```

and you get an answer close to the following (depending on the sample you get)

```
Estimated parameters:
-0.98133      1.0056
- sum of log-likelihoods): -8780.5937
```

Some explanations here. In the name-value pair `'reconst_fraction', [10 0.01]`, we specify our intention to reconstruct a stratified random sample using 10 strata, representing just 1% of the entire data. Note that, here we consider stratification of time points (1,2, 3, …) rather than that of data values. The accuracy of the estimated parameters is good (bearing in mind that the true solution is `[-1,1]`). *It is crucial to realize that the resolutions of the mother dataset and its random sample*. Both datasets maintain the same resolution; only their lengths differ (which is why we set `dt = 0.01`, matching the resolution of the OU mother dataset). Given the high resolution of our data, further Hermite reconstruction may not significantly improve the results. Below are the complete code lines for implementing both Euler and Hermite reconstructions.

```
S = load('OUdata1D.mat');data = S.data;
dt = 0.01
mu = @(x,par)par(1).*x;sigma = @(x,par)par(2);
result16 = euler_reconstruction(data, dt, 'mu', mu, 'sigma', sigma, 'gradient_fun',
eulergrad(mu, sigma), ...
    'reconst_fraction', [10 0.01], 'lb', [-200 eps], 'ub', [200
200],'useparallel',true, 'solver', 'fmincon', 'search_agents', 5);
legpoints16 = legitimate_points(data, dt, 'prev', result16, 'prev_range', 0.5, 'j',
3, 'k', 4);
result_her16 = hermite_reconstruction(data, dt, 'prev', legpoints16, 'solver',
'fmincon');
```

17. Handling replicate datasets

Before proceeding, ensure that you have added the path of the 'Burg' folder to your MATLAB working directory (refer to subsection 6.2 for detailed instructions).

In some cases, we may not have access to a single long dataset but rather to many shorter samples, known as '*replicate data*'. Reconstructing such data is not challenging as long as there is sufficient evidence or theoretical justification to believe that all the data share a common generating system. It's important to note that diffusion models are 'Markov' models, meaning that the future state, given the present state, is independent of the entire past history of states. Therefore, any damage to data at a single time point will only affect the adjacent data points, allowing us to effectively treat damaged values as missing values (`NaN`). Consequently, we can safely append a `NaN` at the end of each replicate and then concatenate all the replicates (the order of concatenation does not matter) to create a long dataset. The code '`prepare_replicateData.m`' automates this process. Once the replicate data is prepared, the subsequent calculations are straightforward. You can simply apply the same codes developed for 'typical' datasets (i.e., single time series datasets) to the replicate data. Note that the replicate data must be supplied as a cell array.

Example 1. In this analysis, we examine a dataset comprising three high-resolution replicates simulated from the grazing model of May, with parameters matching those in **Example 1**. Each replicate begins from the initial state $x_0 =$ 8 and continues until perturbations drive the system towards 0 biomass. To ensure the removal of transient effects, the first 5% of each replicate is discarded. Subsequently, we reconstruct this high-resolution replicate dataset using cubic splines as below

```
S = load('MayData1D_Replicate.mat');
data = S.data; % replicate data should be supplied as a cell array
data = prepare_replicateData(data);% to reconstruct replicate data we first need to
use this function. The rest of calculations are similar to those for typical datasets
L = 0;
R = max(data);
```

```
dt = 0.1;  % this is the true resolution of replicates
mu = 8; sigma = 8;
result17 = euler_reconstruction(data, dt, 'nKnots', [mu sigma], 'spline', 'CC', 'L',
L, 'R', R, ...,
'lb', [zeros(1, mu) - 10, zeros(1, sigma)+eps], 'ub', zeros(1, mu + sigma) + 10,
'solver', 'fmincon', 'search_agents', 5);
```

Which leads us to the following solution

```
Estimated parameters:
0.25467    0.079432  -0.046699    -0.0092472    0.088873  0.0069646  -0.24373    -0.31535
0.47003   0.39488    0.3922     0.40622     0.40369     0.39681     0.38765     0.46541
- sum of log-likelihoods): -9192.4963
```

and for a plot type (see **Figure 14**)

```
r=1.01;K=10;g=2.75;a=1.6;s=0.4;  % true parameter values
par = [r K g a s];
mu = @(x,par)r.*x.*(1-x./K)-g.*x.^2./(x.^2+a.^2);sigma=@(x,par)s;  % true model
xplot=linspace(L,R,2000);
plot_results(result17,xplot,mu(xplot,par),sigma(xplot,par));
```

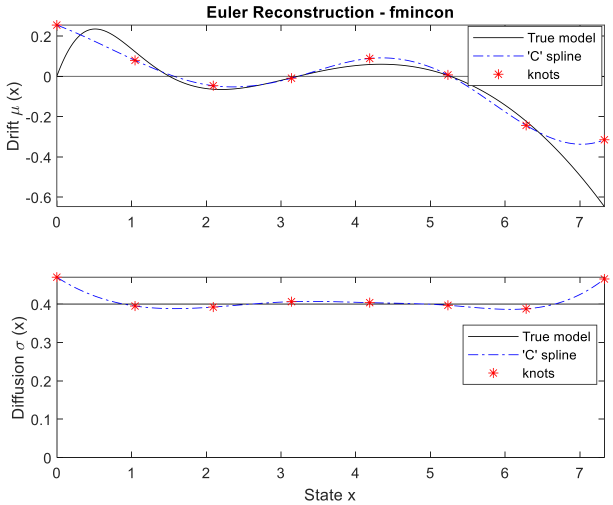


Figure 14. Reconstructing a replicate dataset. The top and bottom panels depict the true drift and diffusion functions (solid black curves) alongside the estimated drift and diffusion functions (dot-dashed blue curves) obtained through spline modeling. The dataset consists of three replicates, all simulated from the May model and initiated at position $x_0 = 8$ until reaching 0 biomass. The true model parameters align with those described in **Example 1**.

An explanation for the absence of the left tail in the reconstructed May model

As is evident in **Figure 3** and **Figure 7**, and **Figure 14** the left tail of the grazing model of May did not manifest in the reconstructed model. This discrepancy between the reconstructed and true models is not attributable to estimation inaccuracies but rather to a deliberate modeling choice in the ecological context. In this model, our objective was to simulate a dataset with positive state values (biomass), despite the stochastic force potentially pushing trajectories into negative states. To address this, we implemented a ‘reflecting’ boundary at `0` biomass, effectively pushing trajectories back to positive values upon crossing `0`. Consequently, the reconstructed model exhibits a steep positive rate of change at `0`, in contrast to May's deterministic model where trajectories slow down near `0` (note that `0` is an equilibrium in the determinist May model). As a result, no positive dataset can reflect this behavior, causing the left tail of May's model to be omitted—a feature that holds limited ecological significance. To reconstruct the left tail, trajectories need to be allowed to cross `0` and fluctuate around it. To illustrate this concept, we have generated a replicate dataset

'`MayData_LeftTail.mat`' which can reveal the left tail of May's model. This dataset consists of 15 replicates, all initially placed at $x_0 = 0.2$ and terminate once they escape the interval [-1 7] via either of the left or right borders (or, they reach the chosen maximum length of 2×10^4). To exclude transient effects the first 5% of all the replicates are discarded. This dataset has a high-resolution. Therefore, we apply Euler reconstruction and fit a cubic spline model. Type the following commands to recover the left tail in the May model (see **Figure 15**)

```
S = load('MayData_LeftTail.mat');
data = S.data; % replicate data should be supplied as a cell array
data = prepare_replicateData(data);
L = min(data); % note that here the min data value is negative (-0.9493)
R = max(data);
dt = 0.1;  % This is the actual time step used to generate this dataset
mu = 8; sigma = 8;
result18 = euler_reconstruction(data, dt, 'nKnots', [mu sigma], 'spline', 'CC', 'L',
L, 'R', R, ...,
'lb', [zeros(1, mu) - 10, zeros(1, sigma)+eps], 'ub', zeros(1, mu + sigma) + 10,
'solver', 'fmincon', 'search_agents', 5);

r = 1.01;K = 10;g = 2.75;a = 1.6;s = 0.4;  % true parameter values
par = [r K g a s];
mu = @(x,par)r.*x.*(1-x./K)-g.*x.^2./(x.^2+a.^2);sigma=@(x,par)s;  % true model
xplot = linspace(L,R,2000);
plot_results(result18,xplot,mu(xplot,par),sigma(xplot,par));
```

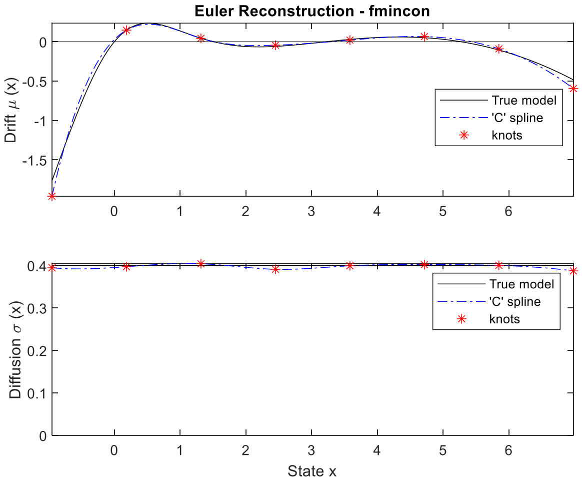


Figure 15. Revealing the left tail in the May model. Estimated drift (top panel) and diffusion (bottom panel) functions using a dataset with negative values. In order to recover the left tail of the May model a dataset with negative biomass is needed (which is ecologically unrealistic). The dataset consists of three replicates all initialized at $x_0 = 0.2$ and terminate once they escape the interval [-1 7] via either of the left or right borders (or, they reach the chosen maximum length of 2×10^4). The first 5% of all replicates are discarded to remove the transient effects.

18. Assessing the uncertainty of the results

After estimating the model parameters, we further need to have an estimate about the uncertainty of the estimated parameters. To this end, we need to use the code 'Uncertainty.m'. Here, we have made one single code which is responsible to calculate the uncertainty of the estimated model parameters for all different types of models (parametric, spline), different reconstruction schemes (Euler, Hermite), different data types (typical, replicate, big), and, with or without missing values. Here, we estimate the uncertainty of the parameters for several examples in this tutorial.

Consider **Example 1**. To assess the uncertainty of the parameters, type the following command lines

```
syms mu(x) sigma(x)
par = sym('par%d', [1 2]);
mu(x) = par(1)*x;sigma(x) = par(2);
S=load('OUdata1D.mat');data=S.data;dt=0.01;
ModelType=["Parametric" "Euler"];
h = 10^(-3);
estimated_par = [-1.0586 0.99531];
[hess,err_hess,err_par]=Uncertainty(ModelType,estimated_par,h,data,[],[],dt,par,mu,si
gma);
```

and you get

```
Uncertainty of the parameters (in terms of standard deviation)
0.014136   0.00069961
```

References

2004. High-resolution record of Northern Hemisphere climate extending into the last interglacial period. Nature **431**:147-151.

Aït-Sahalia, Y. J. E. 2002. Maximum likelihood estimation of discretely sampled diffusions: a closed-form approximation approach. **70**:223-262.

Arani, B. M., S. R. Carpenter, L. Lahti, E. H. Van Nes, and M. Scheffer. 2021. Exit time as a measure of ecological resilience. Science **372**:eaay4895.

Arani, B. M. S. 2019. Inferring ecosystem states and quantifying their resilience: linking theories to ecological data. Wageningen University

Bakshi, G., and N. J. T. J. o. B. Ju. 2005. A Refinement to Aït-Sahalia's (2002)"Maximum Likelihood Estimation of Discretely Sampled Diffusions: A Closed-Form Approximation Approach". **78**:2037-2052.

Broersen, P. M. 2003. Automatic Time Series Identification Spectral Analysis with MATLAB Toolbox ARMASA. IFAC Proceedings Volumes **36**:1435-1440.

Carpenter, S. R., B. M. Arani, P. C. Hanson, M. Scheffer, E. H. Stanley, and E. Van Nes. 2020. Stochastic dynamics of Cyanobacteria in long-term high-frequency observations of a eutrophic lake. Limnology and Oceanography Letters **5**:331-336.

Dansgaard, W., S. J. Johnsen, H. B. Clausen, D. Dahl-Jensen, N. S. Gundestrup, C. U. Hammer, C. S. Hvidberg, J. P. Steffensen, A. Sveinbjörnsdottir, and J. Jouzel. 1993. Evidence for general instability of past climate from a 250-kyr ice-core record. Nature **364**:218-220.

Dickey, D. A., and W. A. J. J. o. t. A. s. a. Fuller. 1979. Distribution of the estimators for autoregressive time series with a unit root. **74**:427-431.

Einstein, A. J. I. o. t. t. o. t. B. m. 1905. On the movement of small particles suspended in a stationary liquid demanded by the molecular-kinetic theory of heat (English translation, 1956).

Erkelens, J. S., A. Tejada, and A. J. den Dekker. 2013. Identification of Time Series Models From Segments—Application to Scanning Transmission Electron Microscopy Images. IEEE Transactions on Instrumentation and Measurement **62**:3231-3242.

Friedrich, R., J. Peinke, M. Sahimi, and M. R. R. J. P. R. Tabar. 2011. Approaching complexity by stochastic methods: From biological systems to turbulence. **506**:87-162.

Koziel, S., and L. Leifsson. 2013. Surrogate-based modeling and optimization. Springer.

Kwasniok, F., and G. Lohmann. 2009. Deriving dynamical models from paleoclimatic records: Application to glacial millennial-scale climate variability. Physical Review E **80**:066104.

Magnuson, J. J., S. R. Carpenter, and E. H. Stanley. 2023. North Temperate Lakes LTER: High Frequency Data: Meteorological, Dissolved Oxygen, Chlorophyll, Phycocyanin-Lake Mendota Buoy 2006-current.

May, R. M. 1977. Thresholds and breakpoints in ecosystems with a multiplicity of stable states. Nature **269**:471-477.

Mirjalili, S., S. M. Mirjalili, and A. J. A. i. e. s. Lewis. 2014. Grey wolf optimizer. **69**:46-61.

MS Arani, B., S. R. Carpenter, E. H. van Nes, I. A. van de Leemput, C. Xu, P. G. Lind, and M. Scheffer. 2024. Stochastic regimes can hide the attractors in data, reconstruction algorithms can reveal them. bioRxiv:2024.2002. 2017.580797.

Nadimi-Shahraki, M. H., S. Taghian, and S. Mirjalili. 2021. An improved grey wolf optimizer for solving engineering problems. Expert Systems with Applications **166**:113917.

Rinn, P., P. G. Lind, M. Wächter, and J. J. a. p. a. Peinke. 2016. The Langevin Approach: An R Package for Modeling Markov Processes.

Siegert, S., and R. J. P. R. E. Friedrich. 2001. Modeling of nonlinear Lévy processes by data analysis. **64**:041107.