Journal of Statistical Software

January 2023, Volume 105, Issue 4. doi: 10.18637/jss.v105.i04

jumpdiff: A Python Library for Statistical Inference of Jump-Diffusion Processes in Observational or Experimental Data Sets

Leonardo Rydin Gorjão
NMBU,
Forschungszentrum Jülich,
University of Cologne

Dirk Witthaut
Forschungszentrum Jülich,
University of Cologne

Pedro G. Lind
Oslo Research Center for AI,
NordSTAR,
OsloMet

Abstract

We introduce a Python library, called **jumpdiff**, which includes all necessary functions to assess jump-diffusion processes. This library includes functions which compute a set of non-parametric estimators of all contributions composing a jump-diffusion process, namely the drift, the diffusion, and the stochastic jump strengths. Having a set of measurements from a jump-diffusion process, **jumpdiff** is able to retrieve the evolution equation producing data series statistically equivalent to the series of measurements. The back-end calculations are based on second-order corrections of the conditional moments expressed from the series of Kramers-Moyal coefficients. Additionally, the library is also able to test if stochastic jump contributions are present in the dynamics underlying a set of measurements. Finally, we introduce a simple iterative method for deriving second-order corrections of any Kramers-Moyal coefficient.

Keywords: stochastic differential equations, jump-diffusion processes, Kramers-Moyal expansion, Kramers-Moyal coefficients, Python.

1. Introduction

Models of complex systems based on stochastic processes are ubiquitous across several research fields. However, the complexity of the stochastic contributions lays often beyond pure diffusive processes, such as Brownian motion and random walks, and might include contributions from discontinuous jumps (Pascucci 2011). Jump-diffusion processes incorporate both diffusive contributions as well as stochastic jumps, making them a natural extension of pure diffusive processes (Aït-Sahalia 2004). These processes are well suited to describe data which exhibits contributions from both continuous and discontinuous stochastic noise. However, there is a

lack of reliable computational libraries oriented at estimating all necessary parameters that quantitatively describe jump-diffusion processes. The problem of deriving jump-diffusion equations has been scarcely addressed (Aït-Sahalia and Lo 1998; Friedrich, Peinke, Sahimi, and Tabar 2011; Anvari, Tabar, Peinke, and Lehnertz 2016b). Moreover, proper numerical routines which cover all different kinds of parameteric contributions are still lacking. The main aim of this paper is to introduce and describe **jumpdiff**, a Python (Van Rossum *et al.* 2011) library, which is able to derive the evolution equation governing specific series of measurements from a jump-diffusion process without any *a priori* knowledge of the process.

In recent years, with the goal of going beyond the narrow scope of Gaussian processes, extensions of classical stochastic processes have included jumps, both with the practical aim of better describing the natural world, or because of the riddling mathematical hardship and the interesting results obtained. As a direct extension of the classical Langevin equation, jump-diffusion processes came into focus via the Kramers-Moyal equation (Anvari *et al.* 2016b; Rydin Gorjão, Heysel, Lehnertz, and Tabar 2019). This family of processes embodies both a conventional Gaussian (diffusion) contribution as well as Poissonian jump contributions. From a time series analysis point-of-view, jump-diffusion processes are particularly hard to analyze and have required a revised interpretation under the classic Kramers-Moyal expansion (Kramers 1940; Moyal 1949). This hardship is given by the simultaneous presence of jumps and diffusion and required a revised interpretation under the classic Kramers-Moyal expansion, particularly when the goal is to extract separately the continuous and the discontinuous contributions from a data set. Jump-diffusion processes have found application in mathematical finance, particularly option pricing (Duffie, Pan, and Singleton 2000; Andersen, Benzoni, and Lund 2002; Johannes 2004; Aït-Sahalia and Jacod 2009), electricity markets (Cartea and Figueroa 2005), early-warning signal identification (Dakos *et al.* 2012), soil moisture dynamics (Daly and Porporato 2006), solar radiation and EEG recordings (Anvari *et al.* 2016a,b; Lehnertz, Zabawa, and Tabar 2018), and neural activity (Giraudo and Sacerdote 1997), amongst others.

In this paper we introduce the Python library **jumpdiff**, oriented towards non-parametric estimation of jump-diffusion processes via the Kramers-Moyal equation, which we will discuss shortly. While being a new library, it incorporates some of the recent theoretical and numerical achievements in stochastic methods (Anvari *et al.* 2016b; Rydin Gorjão *et al.* 2019; Tabar 2019). For example, Langevin processes have been implemented in an R (R Core Team 2022) package and an Python library, cf. Rinn, Lind, Wächter, and Peinke (2016) and Rydin Gorjão and Meirinhos (2019), respectively, enabling to model time series with drift and diffusion strengths. These packages can similarly non-parametrically estimate the strength of drift, diffusion, and other elements in data, but do not close the gap of directly evaluating and extracting the parameters of the analyzed time series. Other packages, such as Julia's (Bezanson, Edelman, Karpinski, and Shah 2017) **DiffEqJump.jl** (Rackauckas and Nie 2017, see `https://github.com/SciML/DiffEqJump.jl`), MATLAB's (The MathWorks Inc. 2021) **PROJ_Option_Pricing_Matlab** (Kirkby, Nguyen, Cui, Zhang, Deng, and Aguilar 2021), or C++'s **DerivativesPricing** (Gosain 2020) focus primarily on generating time series. Library **jumpdiff** allows not only to generate a series of values from a synthetic time series but also to recover all terms composing the evolution equation governing empirical sets of measurements from real jump-diffusion processes.

For the implementation of the **jumpdiff** library, we address two aspects in this paper. First, at the theoretical level, we introduce a set of higher-order finite-time corrections to the Kramers-

Moyal equation, which are particularly relevant for non-parametric estimation of jumps, and in the presence of measurement noise (Böttcher, Peinke, Kleinhans, Friedrich, Lind, and Haase 2006; Lehle 2011, 2013; Scholz *et al.* 2017). Second, at the numerical level, we implement a kernel density Nadaraya-Watson estimator (Nadaraya 1964; Watson 1964) to calculate the Kramers-Moyal coefficients, employing the aforementioned developed finite-time higher-order corrections in the library, and design a simple method to extract the jump contributions. Library **jumpdiff** enables the extraction of the parameters of the system in a non-parametric manner, as well as the distinction between pure diffusions and jump-diffusions. The library is suited for the family of Poissonian jump processes and represents a step forward towards more complex stochastic processes, namely Lévy-like processes (Siegert and Friedrich 2001; Lubashevsky, Friedrich, and Heuer 2009; Zaburdaev, Denisov, and Klafter 2015; Zan, Xu, Kurths, Chechkin, and Metzler 2020).

The paper is organized as follows: In Section 2 we introduce the main theoretical background behind jump-diffusion processes and outline the numerical approximation of Kramers-Moyal coefficients included in the **jumpdiff** library. In Section 3 a full description of all functions composing the library is presented, as well as a simple how-to-use guide. Section 4 concludes the paper, putting some aspects in perspective. Readers familiar with the theory and interested in the implementation and usage of the library **jumpdiff** only may directly go to Section 3. A more descriptive and hands-on explanation can be found in the documentation of the library at `https://jumpdiff.readthedocs.io/`, and the repository of the library is hosted at `https://github.com/LRydin/jumpdiff`.

2. Evolution and inference of jump-diffusion processes

In the following we introduce the stochastic differential equation describing a jump-diffusion process. We describe it together with its counterpart, the Kramers-Moyal equation, a partial differential equation that generalizes Langevin processes and can include discontinuous elements. Then we briefly explain how the terms, so-called coefficients, in the Kramers-Moyal equation link to the parameters of the jump-diffusion process. This is central for two reasons: First, the non-parametric technique employed in **jumpdiff** is based on the Kramers-Moyal equation. Second, we present a set of corrective terms to the estimation of the parameters via the Kramers-Moyal equation, which we include in **jumpdiff** to improve the estimation quality of the parameters.

2.1. Theoretical aspects behind jump-diffusion processes

In this section we consider the stochastic evolution of a time-continuous Markov process, $X(t) \in \mathbb{R}$, that is governed by three independent contributions: one drift strength, one diffusive strength, and one Poissonian (jump) strength. The evolution equation of such a variable reads:

$$\mathrm{d}X(t) = a(x,t)\ \mathrm{d}t + b(x,t)\ \mathrm{d}W(t) + \xi \mathrm{d}J(t), \tag{1}$$

where $a(x,t)$ is the drift strength, $b(x,t)$ is the diffusion or volatility, $W(t)$ is a Wiener process, and $J(t)$ is a time-homogeneous Poisson jump process with rate $\lambda(x,t)$ and an amplitude ξ, which is normally distributed as $\xi \sim \mathcal{N}(0, \sigma_\xi^2)$.

Jump-diffusion processes governed by (1) are a generalization of diffusion processes, since one recovers the latter when $\sigma_\xi = 0$ (similarly $\lambda(x,t) = 0$). Below, we make use of the

Kramers-Moyal expansion to connect the orders of the expansion with each parameter of the jump-diffusion process, thus enabling their estimation.

In order to link (1) to the evolution of the probability $p(x, t+\tau|x', t)$, we employ the Kramers-Moyal equation

$$\frac{\partial}{\partial\tau}p(x, t+\tau|x', t) = \mathcal{L}_{\mathrm{KM}}p(x, t+\tau|x', t), \tag{2}$$

where $\mathcal{L}_{\mathrm{KM}}$ denotes the Kramers-Moyal operator defined as (Risken 1996)

$$\mathcal{L}_{\mathrm{KM}} = \sum_{m=1}^{\infty}\left(-\frac{\partial}{\partial x}\right)^m D_m(x). \tag{3}$$

Functions $D_m(x)$, called Kramers-Moyal coefficients, relate to the m-order conditional moment $M_m(x, \tau)$, given by

$$M_m(x, t, \tau) = \int_{-\infty}^{\infty}(x' - x)^m p(x', t+\tau|x, t)\,\mathrm{d}x'. \tag{4}$$

For simplicity we drop the t-dependency focusing on stationary processes. The Kramers-Moyal coefficients $D_m(x)$ are thus defined for any integer m as

$$D_m(x) = \frac{1}{m!}\lim_{\tau\to 0}\frac{M_m(x, \tau)}{\tau}\,. \tag{5}$$

The jump-diffusion process defined in (1) is linked to a particular case of the Kramers-Moyal expansion defined in (2) and (3), namely

$$\frac{\partial}{\partial\tau}p(x, t+\tau|x', t) = \left[-a(x, t)\frac{\partial}{\partial x} + \Big(b(x, t)^2 + \lambda(x, t)\sigma_\xi\Big)\frac{\partial^2}{\partial x^2} + \sum_{k=2}^{\infty}\sigma_\xi^k\lambda\frac{\partial^{2k}}{\partial x^{2k}}\right]p(x, t+\tau|x', t)\,. \tag{6}$$

With this, we can invert the problem and ask instead the question: If we have a single realization of a stochastic process $X(t)$, can we use the estimation of the Kramers-Moyal coefficients to uncover the parameters of the analyzed realization? The answer for jump-diffusion processes is straightforward, and the parameters defining (1) are given by:

$$a(x, t) = D_1(x, t), \tag{7a}$$

$$b^2(x, t) = D_2(x, t) - \lambda(x, t)\sigma_\xi^2, \tag{7b}$$

$$\sigma_\xi^2 = \frac{D_6(x, t)}{5D_4(x, t)}, \tag{7c}$$

$$\lambda(x, t) = \frac{D_4(x, t)}{3\sigma_\xi^4}. \tag{7d}$$

Notice the relation of the parameters of the jump elements in (1) to the Kramers-Moyal coefficients in (5) is given by higher-order terms $D_{2m} = (2m!)\,\sigma_\xi^m\,\lambda$, for $m \geq 3$ (Anvari *et al.* 2016b).

With this at hand, we can equate that estimating the drift strength $a(x)$, the diffusion strength $b(x)$, and the jump element ξ, translates to estimating the Kramers-Moyal coefficients. The job now is to provide an accurate estimation and to perform the inversion from Kramers-Moyal coefficients to the aforementioned parameters, which is the task of library **jumpdiff**.

It is important to notice that for purely diffusive processes, (2) reduces to the Fokker-Planck-Kolmogorov equation (sometimes denoted solely Fokker-Planck or Smoluchowski equation; Risken 1996), and the estimates of the two first Kramers-Moyal coefficients are sufficient to fully describe a diffusive process. In the case of jump-diffusion processes, higher-order Kramers-Moyal coefficients need to be taken into account as they are non-vanishing. For instance, for the jump-diffusion processes in (1), one can invert the problem from Kramers-Moyal coefficients to the stochastic parameters knowing the first six Kramers-Moyal coefficients (Anvari *et al.* 2016b).

Moreover, as we shall detail in the subsequent section, up to now, higher-order Kramers-Moyal coefficients were estimated from first-order estimations of the conditional moments. Here, we derive the second-order approximations, which imply a more cumbersome analytical approach to the Kramers-Moyal equation 2. These second-order approximations are crucial to improve the estimation of jump-diffusion processes.

2.2. Numerical computation of the Kramers-Moyal coefficients

The numerical computation of the Kramers-Moyal coefficients is based on the numerical computation of conditional moments $M_m(x,t)$, which can be estimated directly from a set of measurements $X(t)$. Indeed, the instantaneous time rate of the moment of order m for the process $X(t)$, conditioned to a specific value x, is given by

$$M_m(x,\tau) = \langle (X(t+\tau) - X(t))^m | X(t) = x \rangle \,, \tag{8}$$

with $\langle X(t) \rangle$ denoting the average of $X(t)$, for all measured t.

The conditional moments can be expressed as sums of products of Kramers-Moyal coefficients, derived from the formal solution of (2), namely

$$p(x,t+\tau|x',t) = \exp\left(\tau \mathcal{L}_{\mathrm{KM}}\right)\delta(x-x') = \sum_{k=0}^{\infty} \frac{(\tau \mathcal{L}_{\mathrm{KM}})^k}{k!}\delta(x-x') \,. \tag{9}$$

Depending on the number of terms used from the sum in (9) one obtains different orders of approximation of the Kramers-Moyal operator. Here we will consider first- and second-order approximations.

First-order approximation

The first-order approximation is given by

$$\exp\left(\tau \mathcal{L}_{\mathrm{KM}}\right) \sim 1 + \tau \mathcal{L}_{\mathrm{KM}} \,, \tag{10}$$

where expressing the Kramers-Moyal coefficients from (5), via the moments in (4), yields

$$D_m(x) = \frac{1}{m!} \lim_{\tau \to 0} \frac{M_m(x,\tau)}{\tau}. \tag{11}$$

The full derivation is given in Appendix A. The derivation is not cumbersome and well known to the community[1], thus of little interest to show here.

[1]We point the reader to a textbook on stochastic processes, e.g., Risken (1996); Gardiner (2009); Tabar (2019).

Second-order approximation

Higher-order approximations of the conditional moment are especially relevant when handling low-sampled data. The second-order approximation of the conditional moments takes in one additional term from the sum in (9), namely

$$\exp\left(\tau\mathcal{L}_{\mathrm{KM}}\right) \sim 1 + \tau\mathcal{L}_{\mathrm{KM}} + \frac{\tau^2}{2}\mathcal{L}_{\mathrm{KM}}\mathcal{L}_{\mathrm{KM}}, \tag{12}$$

where naturally the first-order terms $\sim \tau$ are present at the first-order of the approximation. The derivation, found in full detail in Appendix A, involves a set of relations between the moments $M_m(x,\tau)$ of a given order m and the Kramers-Moyal coefficients $D_n(x)$ of all orders $0 < n \leq m$. Inverting these relations with respect to the Kramers-Moyal coefficients yields

$$D_m(x) = \frac{1}{m!}\lim_{\tau\to 0}\frac{F_m(x,\tau)}{\tau}\,, \tag{13}$$

where $F_m(x,\tau)$ denotes the second-order approximation, in comparison with the first-order approximation given above in (11). The second-order approximations $F_m(x,\tau)$ are given by (dependencies removed for clarity)

$$F_1 = M_1, \tag{14a}$$
$$F_2 = M_2 - M_1^2, \tag{14b}$$
$$F_3 = M_3 - 3M_1M_2 + 3M_1^3, \tag{14c}$$
$$F_4 = M_4 - 4M_1M_3 + 18M_1^2M_2 - 3M_2^2 - 15M_1^4, \tag{14d}$$
$$F_5 = M_5 - 5M_1M_4 + 30M_1^2M_3 - 150M_1^3M_2 + 45M_1M_2^2 - 10M_2M_3 + 105M_1^5, \tag{14e}$$
$$F_6 = M_6 - 6M_1M_5 + 45M_1^2M_4 - 300M_1^3M_3 + 1575M_1^4M_2 - 675M_1^2M_2^2 + 180M_1M_2M_3 + 45M_2^3 - 15M_2M_4 - 10M_3^2 - 945M_1^6\,. \tag{14f}$$

Here naturally the first term on each right-hand side is the first-order approximation. This second-order approximation neglects terms including derivatives of the Kramers-Moyal coefficients, which enables one to express the nth Kramers-Moyal coefficient as a function of conditional moments up to order $n-1$. In this way, we provide a general formula for improving the estimates of Kramers-Moyal coefficient, taken as linear approximations of the corresponding conditional moment. The full derivation is given in Appendix A.

3. Implementation of the functions in library jumpdiff

In this section we introduce **jumpdiff** and some of its main functions and utilities. The simplest way to install **jumpdiff** is via the Python Package Index (PyPI), simply using

```
$ pip install jumpdiff
```

or with the readers' favorite Python package installation program. The library has three main functionalities for addressing jump-diffusion processes, namely:

- Generate sample trajectories of jump-diffusion processes.

Function	Parameters	Outputs	Internal libraries	External libraries
`jd_process()`	`time`, `delta_t`, drift `a`, diffusion `b`, jump amplitude `xi`, jump rate `lamb`	timeseries `X`	None	**numpy**
`moments()`	`timeseries`, bins `bins`, order `power`, time lag `lag`, `correction`	space `edges`, `moments`	**binning**, **kernels**	**numpy**, **scipy**
`jump_amplitude()`	`moments`	estimator `xi_est`	None	**numpy**
`jump_rate()`	`moments`, jump amplitude `xi_est`	estimator `lamb_est`	None	**numpy**
`q_ratio()`	time lag `lag`, `timeseries`	time lag `lag`, ratio `ratio`	**moments**	**numpy**
`corrections()`	moments `m`, order `power`	corrected `moments`	None	**numpy**
`m_formula()`	order `power`	symbolic `term`	None	**sympy**
`f_formula()`	order `power`	symbolic `term`	None	**sympy**

Table 1: Primary functions (top) and helping functions (bottom) implemented in the **jumpdiff** library.

- Extract all parameters for a single trajectory of a jump-diffusion process.
- Evaluate if a trajectory is a purely diffusion process or a jump-diffusion process.

We note here that the user interested in evaluating their own data should start with the last mentioned step, i.e., to firstly evaluate if their data is described by a jump-diffusion process.

In this section, we describe in detail each one of these functionalities, explaining how we employ the most important functions in **jumpdiff** to estimate the parameters of a jump-diffusion process. All functions included in library **jumpdiff** are listed in Table 1.

The library **jumpdiff** requires the staple Python libraries **numpy** (Van der Walt, Colbert, and Varoquaux 2011), **scipy** (Virtanen *et al.* 2020), and **sympy** (Meurer *et al.* 2017), and a Python version $\geq$ 3.4.

3.1. From equation to data: Generating sample trajectories

To start off, import the library into a working Python environment, using

```
>>> import jumpdiff as jd
```

Subsequently, all functions of **jumpdiff** can be access via `jd`. To generate a sample trajectory of a jump-diffusion process, call the function `jd_process()`. We will generate a single trajectory of the process, and subsequently employ the non-parametric estimators in **jumpdiff** to retrieve the parameters of the jump-diffusion process generated, described in the following subsections.

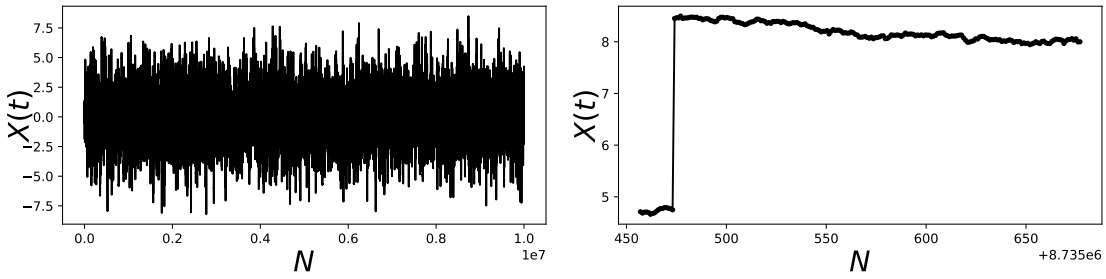

Figure 1: Illustration of a jump-diffusion process using function `jd_process()` which implements (15). (Left) the full extent of (15) with $N = 10^7$. (Right) an exemplary jump in the integrated process. Here $\theta = 0.5$, $\sigma = 0.75$, $\sigma_\xi^2 = 1.5$, and $\lambda = 1.75$.

Let us take a simple example and generate a trajectory of an Ornstein-Uhlenbeck process with Poissonian jumps, given by equation

$$\mathrm{d}X = -\theta x \, \mathrm{d}t + \sigma \, \mathrm{d}W(t) + \xi \, \mathrm{d}J(t) \, . \tag{15}$$

Naturally we also need to specify the values of θ, the drift strength, σ, the diffusion strength, and σ_ξ and λ, the standard deviation of ξ and the jump rate of $J(t)$. We can use `jd_process()` for integrating a jump-diffusion process, with a number of points $N = 1 \times 10^7$ ($t = 10^4$) and a time-step of $\Delta t = 0.001$.

This is implemented in the following way: First, specify the integration time and time sampling:

```
>>> time = 10000
>>> delta_t = 0.001
```

Then define the drift function $a(x)$ and the diffusion function $b(x)$:

```
>>> def a(x):
...     return -0.5*x
>>> def b(x):
...     return 0.75
```

Define jump amplitude and rate

```
>>> xi = 1.5
>>> lamb = 1.75
```

and generate the jump-diffusion process:

```
>>> X = jd.jd_process(time, delta_t, a, b, xi, lamb)
```

In Figure 1 we plot the the trajectory generated with this code alongside with a zoomed region where a jump can be distinctly seen.

While in this example an Ornstein-Uhlenbeck process, i.e., a linear drift strength and constant diffusion strength, is chosen with a Poissonian jump strength, other higher polynomial

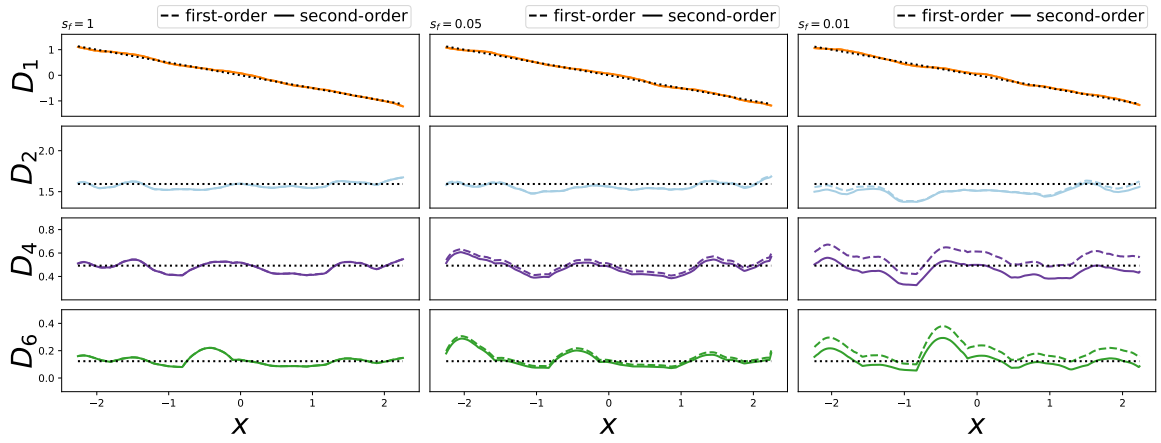

Figure 2: Kramers-Moyal coefficients $D_n(x)$ computed with first-order approximations (dashed lines) and second-order approximations (solid lines). The data set used for computing Kramers-Moyal coefficients was generated by integrating (15) with the same parameter values as in Figure 1. Three sampling rates are consider: (left) every integrated value, $s_f = 1$, (middle) $s_f = 0.05$, (right) $s_f = 0.01$. The dotted line indicates the theoretical result.

functions can be chosen for the different contributions by adjusting the drift and diffusion functions.

3.2. From data to the jump-diffusion equation

Having described how to generate series of values from a jump-diffusion equation, using the function `jd_process()`, we now consider the inverse problem: Starting from that series of values, derive the parameters of the jump-diffusion equation. To that end, we extract the Kramers-Moyal coefficients $D_m(x)$ of this exact simulated process `X`, such that we can employ the non-parametric estimation method and compare if the parameters extracted match the parameters chosen.

We now employ the function `moments()` on the generated time series `X` to extract the Kramers-Moyal coefficients. This we achieve using: First, we extract the Kramers-Moyal coefficients and state-space without second-order corrections:

```
>>> edge, simple_mom = jd.moments(timeseries=X, correction=False)
```

and then with second-order corrections:

```
>>> edge, mom = jd.moments(timeseries=X, correction=True)
```

In the case above we include the estimation of the Kramers-Moyal coefficients both with and without the corrective terms we designed. Figure 2 shows the derived results for the inverse problem, using first-order (dashed lines) and second-order corrections (solid lines). The theoretical values are indicated by the dotted lines.

In order to showcase the importance of the corrective terms (`correction = True`), we show, for the generated process `X`, the estimation of the Kramers-Moyal coefficients while downsampling data. We consider different sampling rates, namely $s_f = 1, 0.05$, and 0.01, i.e.,

taking the full data, only every 20th datapoint, and only every 100th datapoint. As can be seen in Figure 2, the second-order corrective term improves the estimate of all Kramers-Moyal coefficients $D_m(x)$, $m \geq 2$, especially in the cases with lowest sampling rates. We can see in Figure 2 that the function `moments()` has estimated up to six Kramers-Moyal coefficients $D_n(x)$. We will now turn – after a short paragraph – to two specific functions in **jumpdiff** that allow us to directly extract the jump amplitude σ_ξ^2 and the jump rate λ, as these are the contributions from the discontinuous part to the process.

Before we turn to the estimation of the jump elements, note that technically, the function `moments()` performs a kernel-density estimation employing a Nadaraya-Watson estimator (Nadaraya 1964; Watson 1964) of the different orders of the moments. By default it employs an Epanechnikov kernel and uses a convolution method to perform the kernel-density estimation of the moments. Four other kernels are also available, namely Gaussian, uniform, triangular, and quartic kernels. Moreover, the function `moments()` includes the input parameter `lag`, which is a time-lag that enables one to estimate the conditional moments at different time-lags τ. If left unspecified, it assumes the shortest increment of the time series, i.e., the time series sampling rate $\tau = 1/s_f$. This is especially suited for evaluating the conditional moments in the limiting case of $\tau \to 0$, since numerical accuracy is bounded by the sampling rate s_f. This evaluation is done by plotting a few time-lags, e.g., $\tau = 1/s_f, \ldots, 10/s_f$, and extrapolate the limit $\tau \to 0$ (Böttcher *et al.* 2006; Lind, Haase, Böttcher, Peinke, Kleinhans, and Friedrich 2010).

In Figure 2 we can see the estimation of the Kramers-Moyal coefficients, with first- and second-order corrections. The results reproduce well the theoretical values (dotted lines), allowing us to extract the parameters of our process via (7). Retrieving the drift and diffusion functions, $a(x,t)$ and $b(x,t)$ respectively, are known problems, which imply studying the first and second Kramers-Moyal coefficients.

From (7) one recovers also the jump amplitude σ_ξ^2 and the jump rate λ of the time series, by considering higher-order conditional moments. In **jumpdiff**, functions `jump_rate()` and `jump_amplitude()` implement (7d) and (7c), respectively. To estimate the jump amplitude σ_ξ^2 and the jump rate λ of the time series, use

```
>>> xi_est = jd.jump_amplitude(moments=simple_mom)
```

to estimate the jump amplitude and

```
>>> lamb_est = jd.jump_rate(moments=simple_mom)
```

to estimate the jump rate. This yields a numerical estimation of each of the parameters. The user can as well utilize `mom` instead of `simple_mom`.

Here we leave an important note for the user, regarding the impact of a low number of data points in a time series. The estimation of the jump amplitude σ_ξ^2 and the jump rate λ is strongly dependent on the number of data points in each time series. In Figure 3 we numerically integrate (15), as before, and employ the estimators of the jump amplitude and jump rate, i.e., functions `jump_rate()` and `jump_amplitude()`, for a time series with increasing number of data points N. The estimators converge to the theoretical values (dashed line) with increasing accuracy with the average number on jumps $\langle\lambda\rangle$ in the time series. It is central to understand that short time series cause an under-estimation of σ_ξ^2 and simultaneously an

	$a(x)$	$b(x)$	σ_ξ^2	λ
Theoretical	$-0.5x$	0.75	1.5	1.75
Estimated	$-0.496x$	0.760	1.524	1.802

Table 2: Parameter estimation for the process (15) in Figure 1, generated with `jd_process()`, with indicated parameters.

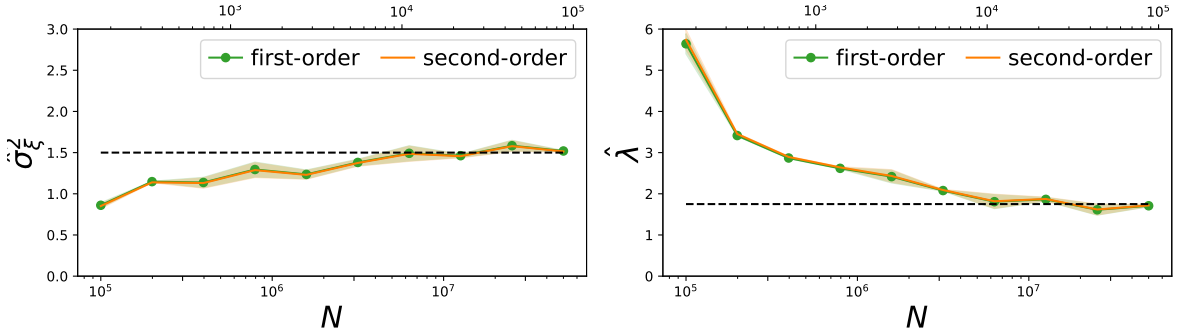

Figure 3: Estimation of the jump amplitude $\hat{\sigma}_\xi^2$ (left) and the jump rate $\hat{\lambda}$ (right) from 20 numerically simulated jump-diffusion processes with increasing time lengths N. The data was generated by integrating (15), with the $\sigma_\xi^2 = 1.5$ and $\lambda = 1.75$ (dashed lines), and each term was estimated using the functions `jump_amplitude()` and `jump_rate()`. The estimates are shown as a function of the number of points $N \in [1 \times 10^5, 5 \times 10^7]$, always with a time-step $\Delta t = 0.001$. Top axis denotes the average number of jumps $\langle\lambda\rangle$ in the respective numerically integrated time series. Each point is an average over 10 iterations. Standard deviations depicted in the shaded areas.

over-estimation of λ. This also springboard us to the next important question: How can we know our time series is a jump-diffusion?

Before we move on, we can compare the theoretical values used in (15) and estimated parameters via fitting the slopes of $D_1(x)$ and the constant values of $D_2(x)$ in Figure 2, as well as the used functions `jump_rate()` and `jump_amplitude()`. The values can be found in Table 2 and agree with our constructed model. For a general method to recover all Kramers-Moyal coefficients, for stochastic processes of any dimension, see Rydin Gorjão and Meirinhos (2019).

3.3. Distinguishing between purely diffusive and jump-diffusion processes

As mentioned at the start of this section, when analyzing real measurements or observational data, the following steps are crucial to understand whether the data at hand fall into the category of a jump-diffusion process or not.

The presence of a discontinuous term in (1) is the fundamental addition to a general diffusion process. However, assuming such *Ansatz* for purely diffusive process may lead to spurious jump terms. To avoid this, one fundamental question to ask is whether we are in the presence of a pure diffusion or a jump-diffusion process, in order to choose *ab initio* the proper *Ansatz*.

To differentiate diffusive from jump-diffusion processes, Lehnertz *et al.* (2018) introduced a simple criterion, which is based on the scaling of the ratio of the fourth- and sixth-order

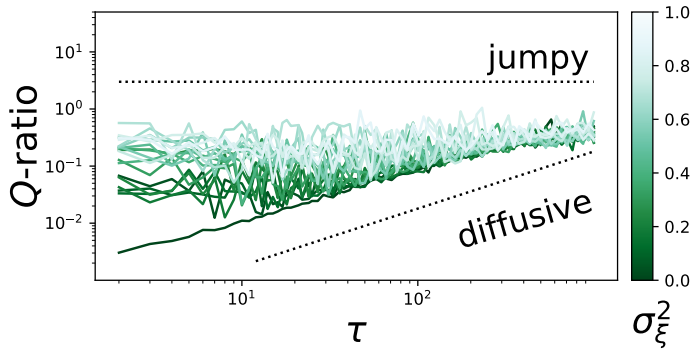

Figure 4: Illustration of the Q-ratio, defined in (16), for the process in (1) with $a(x) = -x$ and $b(x) = 1$ and varying jump amplitudes $\sigma_\xi^2 \in [0, 1]$ with a fixed jump rate $\lambda = 0.1$. The equation was numerically integrated with $t = 1000$ and time step 0.001. Each line is an average of 20 iterations.

conditional moments with the scale τ, denoted the Q-ratio, given by

$$Q(x, \tau) = \frac{M_6(x, \tau)}{5M_4(x, \tau)} \,. \tag{16}$$

If the process is purely diffusive $Q(x, \tau) = \tau(b(x))^2$ (linear function in τ), whereas if the process has a jump term, $Q(x, \tau) = \sigma_\xi^2$ (constant, independent of τ). This criterion can be employed directly for any time series and is implemented as the function `q_ratio()` in **jumpdiff**.

To analyze the scaling of the Q-ratio, follow the recipe: First import **numpy** for generating logarithmic spaced arrays and **matplotlib** for plotting in double logarithmic scale

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
```

Then take a sequence of integer lags

```
>>> lag = np.logspace(0, 4, 25, dtype=int)
```

and recover the Q-ratio of the time series `X` using

```
>>> lag, Q = jd.q_ratio(lag, X)
```

before plotting in a log-log scale

```
>>> plt.loglog(lag, Q)
```

Figure 4 showcases how the Q-ratio changes between purely diffusive to jump-diffusion processes. In this manner, we consider the same process we have introduced in (15), but we will vary the the amplitude of the jumps $\sigma_\xi^2 \in [0, 1]$. Figure 4 illustrates the implementation of the function `q_ratio()`, plotting it for several time-lags τ in a log-log plot. If there is a linear dependence on τ the time series $X(t)$ is a pure diffusive process, whereas if the plot is approximately flat, showing a constant Q-ratio the time series should have a jump term. Notice

that increasing the jump rate from $\xi = 0$ (pure diffusion) to $\xi = b(x) = 1$ (same amplitude as diffusion strength), the Q-ratio changes from linearly depending on τ to a constant. The different relation between the Q-ratio and τ should help the user distinguish which type of data they are analyzing, and thus help assert whether their data follows a purely diffusive process or is better described by a jump-diffusion.

4. Discussion and conclusions

Jump-diffusion processes are stochastic models able to describe processes riddled with jumps. Alongside with regular diffusion processes, their elegance lies in the possibility of estimating, non-parametrically, the parameters via the Kramers-Moyal equation. Access to higher-order moments of the Kramers-Moyal equation is computationally feasible, and thus permits a one-to-one correspondence between model parameters and the estimators (cf. (7)). These, on the other hand, are hampered by the scarcity of jumps. Where the diffusive strengths are ubiquitous, the jumps take place sparsely in time. To this effect, the set of second-order corrections to the Kramers-Moyal operator implemented here are crucial in improving the estimation of the parameters of the model.

The use of jump-diffusion models as descriptives of processes beyond diffusions have seen application across different research fields. The elegance of these processes lies not only in their general applicability, but also in the ease of using non-parametric parameter estimation. In this sense, the presented second-order corrections are of substantial relevance, especially taking into account that jumps occur sparsely. Here we present a two-fold project, developing second-order corrections of the Kramers-Moyal expansion of jump-diffusion processes and designing an easy-to-employ Python library. **jumpdiff** requires minimal mathematical knowledge to be used, and thus can find application even in non-mathematically oriented research fields.

The limitations and finite scope of the library here described motivate further development, namely towards more general jump types, having amplitudes and rates which are time (and space) dependent. Moreover, beyond Poissonian jumps, new libraries can be developed, for instance to approach Lévy-like processes.

Acknowledgments

The authors thank Sílvio M.D. Queirós for useful discussions. L.R.G. and D.W. gratefully acknowledge support by the Helmholtz Association, via the joint initiative *Energy System 2050 – A Contribution of the Research Field Energy*, with grant No. VH-NG-1025, and the associative *Uncertainty Quantification – From Data to Reliable Knowledge (UQ)* with grant No. ZT-I-0029. L.R.G. gratefully acknowledges the scholarship funding from *E.ON Stipendienfonds*, the *STORM – Stochastics for Time-Space Risk Models* project of the Research Council of Norway (RCN) No. 274410, and the Oslo Research Center for AI for partial funding.

References

Aït-Sahalia Y (2004). "Disentangling Diffusion from Jumps." *Journal of Financial Economics*, **74**(3), 487–528. doi:10.1016/j.jfineco.2003.09.005.

Aït-Sahalia Y, Jacod J (2009). "Testing for Jumps in a Discretely Observed Process." *The Annals of Statistics*, **37**(1), 184–222. doi:10.1214/07-AOS568.

Aït-Sahalia Y, Lo AW (1998). "Nonparametric Estimation of State-Price Densities Implicit in Financial Asset Prices." *The Journal of Finance*, **53**(2), 499–547. doi:10.1111/0022-1082.215228.

Andersen TG, Benzoni L, Lund J (2002). "An Empirical Investigation of Continuous-Time Equity Return Models." *The Journal of Finance*, **57**(3), 1239–1284. doi:10.1111/1540-6261.00460.

Anvari M, Lohmann G, Wächter M, Milan P, Lorenz E, Heinemann D, Tabar MRR, Peinke J (2016a). "Short Term Fluctuations of Wind and Solar Power Systems." *New Journal of Physics*, **18**, 063027. doi:10.1088/1367-2630/18/6/063027.

Anvari M, Tabar MRR, Peinke J, Lehnertz K (2016b). "Disentangling the Stochastic Behavior of Complex Time Series." *Scientific Reports*, **6**, 35435. doi:10.1038/srep35435.

Bezanson J, Edelman A, Karpinski S, Shah VB (2017). "Julia: A Fresh Approach to Numerical Computing." *SIAM Review*, **59**(1), 65–98. doi:10.1137/141000671.

Böttcher F, Peinke J, Kleinhans D, Friedrich R, Lind PG, Haase M (2006). "Reconstruction of Complex Dynamical Systems Affected by Strong Measurement Noise." *Physical Review Letters*, **97**, 090603. doi:10.1103/physrevlett.97.090603.

Cartea Á, Figueroa MG (2005). "Pricing in Electricity Markets: A Mean Reverting Jump Diffusion Model with Seasonality." *Applied Mathematical Finance*, **12**(4), 313–335. doi:10.1080/13504860500117503.

Dakos V, Carpenter SR, Brock WA, Ellison AM, Guttal V, Ives AR, Kéfi S, Livina V, Seekell DA, Van Nes EH, Scheffer M (2012). "Methods for Detecting Early Warnings of Critical Transitions in Time Series Illustrated Using Simulated Ecological Data." *PLoS ONE*, **7**(7), 1–20. doi:10.1371/journal.pone.0041010.

Daly E, Porporato A (2006). "Probabilistic Dynamics of Some Jump-Diffusion Systems." *Physical Review E*, **73**, 026108. doi:10.1103/physreve.73.026108.

Duffie D, Pan J, Singleton K (2000). "Transform Analysis and Asset Pricing for Affine Jump-Diffusions." *Econometrica*, **68**(6), 1343–1376. doi:10.1111/1468-0262.00164.

Friedrich R, Peinke J, Sahimi M, Tabar MRR (2011). "Approaching Complexity by Stochastic Methods: From Biological Systems to Turbulence." *Physics Reports*, **506**(5), 87–162. doi:10.1016/j.physrep.2011.05.003.

Gardiner C (2009). *Stochastic Methods: A Handbook for the Natural and Social Sciences*. 4th edition. Springer-Verlag.

Giraudo MT, Sacerdote L (1997). "Jump-Diffusion Processes as Models for Neuronal Activity." *Biosystems*, **40**(1–2), 75–82. doi:10.1016/0303-2647(96)01632-2.

Gosain A (2020). ***DerivativesPricing***: *Derivatives Pricing and Optimisation*. C++ code bundle, URL https://github.com/AnjishtGosain/DerivativesPricing.

Gottschall J, Peinke J (2008). "On the Definition and Handling of Different Drift and Diffusion Estimates." *New Journal of Physics*, **10**, 083034. doi:10.1088/1367-2630/10/8/083034.

Johannes M (2004). "The Statistical and Economic Role of Jumps in Continuous-Time Interest Rate Models." *The Journal of Finance*, **59**(1), 227–260. doi:10.1111/j.1540-6321.2004.00632.x.

Kirkby JL, Nguyen D, Cui Z, Zhang Z, Deng S, Aguilar JP (2021). ***PROJ_Option_Pricing_Matlab**: Option Pricing PROJ Method (Exotic/Vanilla Options)*. MATLAB modules, URL https://github.com/jkirkby3/PROJ_Option_Pricing_Matlab.

Kramers HA (1940). "Brownian Motion in a Field of Force and the Diffusion Model of Chemical Reactions." *Physica*, **7**(4), 284–304. doi:10.1016/s0031-8914(40)90098-2.

Lehle B (2011). "Analysis of Stochastic Time Series in the Presence of Strong Measurement Noise." *Physical Review E*, **83**, 021113. doi:10.1103/physreve.83.021113.

Lehle B (2013). "Stochastic Time Series with Strong, Correlated Measurement Noise: Markov Analysis in N Dimensions." *Journal of Statistical Physics*, **152**, 1145–1169. doi:10.1007/s10955-013-0803-z.

Lehnertz K, Zabawa L, Tabar MRR (2018). "Characterizing Abrupt Transitions in Stochastic Dynamics." *New Journal of Physics*, **20**(11), 113043. doi:10.1088/1367-2630/aaf0d7.

Lind PG, Haase M, Böttcher F, Peinke J, Kleinhans D, Friedrich R (2010). "Extracting Strong Measurement Noise from Stochastic Time Series: Applications to Empirical Data." *Physical Review E*, **81**, 041125. doi:10.1103/physreve.81.041125.

Lubashevsky I, Friedrich R, Heuer A (2009). "Continuous-Time Multidimensional Markovian Description of Lévy Walks." *Physical Review E*, **80**(2), 031148. doi:10.1103/physreve.80.031148.

Meurer A, Smith CP, Paprocki M, Čertík O, Kirpichev SB, Rocklin M, Kumar A, Ivanov S, Moore JK, Singh S, Rathnayake T, Vig S, Granger BE, Muller RP, Bonazzi F, Gupta H, Vats S, Johansson F, Pedregosa F, Curry MJ, Terrel AR, Roučka Š, Saboo A, Fernando I, Kulal S, Cimrman R, Scopatz A (2017). "**SymPy**: Symbolic Computing in Python." *PeerJ Computer Science*, **3**, e103. doi:10.7717/peerj-cs.103.

Moyal JE (1949). "Stochastic Processes and Statistical Physics." *Journal of the Royal Statistical Society B*, **11**(2), 150–210. doi:10.1111/j.2517-6161.1949.tb00030.x.

Nadaraya EA (1964). "On Estimating Regression." *Theory of Probability & Its Applications*, **9**(1), 141–142. doi:10.1137/1109020.

Pascucci A (2011). "Stochastic Calculus for Jump Processes." In *PDE and Martingale Methods in Option Pricing*, pp. 497–540. Springer-Verlag, Milano. doi:10.1007/978-88-470-1781-8_14.

Rackauckas C, Nie Q (2017). "**DifferentialEquations.jl** – A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia." *The Journal of Open Research Software*, **5**(1). doi:10.5334/jors.151.

R Core Team (2022). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Rinn P, Lind PG, Wächter M, Peinke J (2016). "The Langevin Approach: An R Package for Modeling Markov Processes." *Journal of Open Research Software*, **4**, e34. doi:10.5334/jors.123.

Risken H (1996). *The Fokker-Planck Equation: Methods of Solution and Applications*. 2nd edition. Springer-Verlag. doi:10.1007/978-3-642-61544-3.

Rydin Gorjão L, Heysel J, Lehnertz K, Tabar MRR (2019). "Analysis and Data-Driven Reconstruction of Bivariate Jump-Diffusion Processes." *Physical Review E*, **100**, 062127. doi:10.1103/physreve.100.062127.

Rydin Gorjão L, Meirinhos F (2019). "**kramersmoyal**: Kramers-Moyal Coefficients for Stochastic Processes." *Journal of Open Source Software*, **4**(44). doi:10.21105/joss.01693.

Scholz T, Raischel F, Lopes VV, Lehle B, Wächter M, Peinke J, Lind PG (2017). "Parameter-Free Resolution of the Superposition of Stochastic Signals." *Physics Letters A*, **381**(4), 194–206. doi:10.1016/j.physleta.2016.09.057.

Siegert S, Friedrich R (2001). "Modeling of Nonlinear Lévy Processes by Data Analysis." *Physical Review E*, **64**(2), 041107. doi:10.1103/physreve.64.041107.

Tabar MRR (2019). *Analysis and Data-Based Reconstruction of Complex Nonlinear Dynamical Systems*. Springer-Verlag. doi:10.1007/978-3-030-18472-8.

The MathWorks Inc (2021). *MATLAB – The Language of Technical Computing, Version R2021a*. Natick. URL https://www.mathworks.com/products/matlab/.

Van der Walt S, Colbert SC, Varoquaux G (2011). "The **NumPy** Array: A Structure for Efficient Numerical Computation." *Computing in Science Engineering*, **13**(2), 22–30. doi:10.1109/mcse.2011.37.

Van Rossum G, *et al.* (2011). *Python Programming Language*. URL https://www.python.org/.

Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, Burovski E, Peterson P, Weckesser W, Bright J, Van der Walt SJ, Brett M, Wilson J, Millman KJ, Mayorov N, Nelson ARJ, Jones E, Kern R, Larson E, Carey CJ, Polat İ, Feng Y, Moore EW, VanderPlas J, Laxalde D, Perktold J, Cimrman R, Henriksen I, Quintero EA, Harris CR, Archibald AM, Ribeiro AH, Pedregosa F, Van Mulbregt P, **SciPy** 10 Contributors (2020). "**SciPy** 1.0: Fundamental Algorithms for Scientific Computing in Python." *Nature Methods*, **17**, 261–272. doi:10.1038/s41592-019-0686-2.

Watson GS (1964). "Smooth Regression Analysis." *Sankhyā: The Indian Journal of Statistics A*, **26**(4), 359–372.

Zaburdaev V, Denisov S, Klafter J (2015). "Lévy Walks." *Review of Modern Physics*, **87**(2), 483. doi:10.1103/revmodphys.87.483.

Zan W, Xu Y, Kurths J, Chechkin AV, Metzler R (2020). "Stochastic Dynamics Driven by Combined Lévy-Gaussian Noise: Fractional Fokker-Planck-Kolmogorov Equation and Solution." *Journal of Physics A: Mathematical and Theoretical*, **53**(38), 385001. doi:10.1088/1751-8121/aba654.

A. Second-order corrections of Kramers-Moyal coefficients

For the first-order approximation of conditional moments M_n, we substitute in (4) the first-order approximation of the exponential operator, namely

$$\exp\left(\tau\mathcal{L}_{\mathrm{KM}}\right) \sim 1 + \tau\mathcal{L}_{\mathrm{KM}},$$

yielding

$$\begin{aligned} M_n(x',\tau) &\sim \int_{-\infty}^{\infty} (x-x')^n (1+\tau\mathcal{L}_{\mathrm{KM}})\delta(x-x')\,\mathrm{d}x \\ &= \int_{-\infty}^{\infty} (x-x')^n \delta(x-x')\,\mathrm{d}x + \tau\int_{-\infty}^{\infty} (x-x')^n \left[\sum_{m=1}^{\infty}\left(-\frac{\mathrm{d}}{\mathrm{d}x}\right)^m D_m(x)\right]\delta(x-x')\,\mathrm{d}x \\ &= 0 + \tau\sum_{m=1}^{\infty}(-1)^m D_m(x')\int_{-\infty}^{\infty}(x-x')^n\left(\frac{\mathrm{d}}{\mathrm{d}x}\right)^m \delta(x-x')\,\mathrm{d}x. \end{aligned} \tag{17}$$

The last integral is given by:

$$\begin{aligned} I_1 &= \int_{-\infty}^{\infty}(x-x')^n\left(\frac{\mathrm{d}}{\mathrm{d}x}\right)^m\delta(x-x')\,\mathrm{d}x \\ &= \left[(x-x')^n\left(\frac{\mathrm{d}}{\mathrm{d}x}\right)^{m-1}\right]_{-\infty}^{\infty} - \int_{-\infty}^{\infty} n(x-x')^{n-1}\left(\frac{\mathrm{d}}{\mathrm{d}x}\right)^{m-1}\delta(x-x')\,\mathrm{d}x \\ &= 0 - \int_{-\infty}^{\infty} n(x-x')^{n-1}\left(\frac{\mathrm{d}}{\mathrm{d}x}\right)^{m-1}\delta(x-x')\,\mathrm{d}x \\ &= \begin{cases} (-1)^n(n!)\displaystyle\int_{-\infty}^{\infty}\left(\frac{\mathrm{d}}{\mathrm{d}x}\right)^{m-n}\delta(x-x')\,\mathrm{d}x & \Leftarrow \quad m \geq n \\ (-1)^m\dfrac{n!}{(n-m-1)!}\displaystyle\int_{-\infty}^{\infty}(x-x')^{n-m-1}\delta(x-x')\,\mathrm{d}x & \Leftarrow \quad m < n \end{cases} \\ &= (-1)^n(n!)\delta_{nm} \end{aligned}$$

yielding

$$M_n(x',\tau) \sim (n!)\tau D_n(x'). \tag{18}$$

This approximation is the one used in Anvari *et al.* (2016b).

For the second-order approximation of conditional moments M_n, we substitute in (4) the second-order approximation of the exponential operator, namely

$$\exp\left(\tau\mathcal{L}_{\mathrm{KM}}\right) \sim 1 + \tau\mathcal{L}_{\mathrm{KM}} + \frac{\tau^2}{2}\mathcal{L}_{\mathrm{KM}}\mathcal{L}_{\mathrm{KM}}, \tag{19}$$

yielding

$$
\begin{aligned}
M_n(x',\tau) &\sim \int_{-\infty}^{\infty}(x-x')^n\left(1+\tau\mathcal{L}_{\mathrm{KM}}+\frac{\tau^2}{2}\mathcal{L}_{\mathrm{KM}}\mathcal{L}_{\mathrm{KM}}\right)\delta(x-x')\,\mathrm{d}x\\
&= (n!)\tau D_n(x') + \frac{\tau^2}{2}\int_{-\infty}^{\infty}(x-x')^n\mathcal{L}_{\mathrm{KM}}\mathcal{L}_{\mathrm{KM}}\delta(x-x')\,\mathrm{d}x\\
&= (n!)\tau D_n(x') + \frac{\tau^2}{2}\int_{-\infty}^{\infty}(x-x')^n\left[\sum_{p=1}^{\infty}\left(-\frac{\mathrm{d}}{\mathrm{d}x}\right)^p D_p(x)\right]\left[\sum_{m=1}^{\infty}\left(-\frac{\mathrm{d}}{\mathrm{d}x}\right)^m D_m(x)\right]\delta(x-x')\,\mathrm{d}x\\
&= (n!)\tau D_n(x') + \frac{\tau^2}{2}\sum_{p=1}^{\infty}\sum_{m=1}^{\infty}\int_{-\infty}^{\infty}(x-x')^n\left(-\frac{\mathrm{d}}{\mathrm{d}x}\right)^p D_p(x)\left(-\frac{\mathrm{d}}{\mathrm{d}x}\right)^m D_m(x)\delta(x-x')\,\mathrm{d}x\\
&= (n!)\tau D_n(x') + \frac{\tau^2}{2}\sum_{p=1}^{\infty}\sum_{m=1}^{\infty}(-1)^{p+m}D_m(x')\int_{-\infty}^{\infty}(x-x')^n\left(\frac{\mathrm{d}}{\mathrm{d}x}\right)^p D_p(x)\left(\frac{\mathrm{d}}{\mathrm{d}x}\right)^m\delta(x-x')\,\mathrm{d}x.
\end{aligned}
$$

The last integral is derived as follows:

$$
\begin{aligned}
I_2 &= \int_{-\infty}^{\infty}(x-x')^n\left(\frac{\mathrm{d}}{\mathrm{d}x}\right)^p D_p(x)\left(\frac{\mathrm{d}}{\mathrm{d}x}\right)^m\delta(x-x')\,\mathrm{d}x\\
&= \int_{-\infty}^{\infty}(x-x')^n\sum_{s=0}^{p}\binom{p}{s}\left[\left(\frac{\mathrm{d}}{\mathrm{d}x}\right)^{p-s}D_p(x)\right]\left[\left(\frac{\mathrm{d}}{\mathrm{d}x}\right)^{m+s}\delta(x-x')\right]\mathrm{d}x\\
&= \sum_{s=0}^{p}\binom{p}{s}\int_{-\infty}^{\infty}G(x)\left(\frac{\mathrm{d}}{\mathrm{d}x}\right)^{m+s}\delta(x-x')\,\mathrm{d}x
\end{aligned}
\tag{20}
$$

with

$$
G(x) = (x-x')^n\left(\frac{\mathrm{d}}{\mathrm{d}x}\right)^{p-s}D_p(x).
$$

The last member of (20) is computed in a similar way as integral I_1, yielding

$$
\begin{aligned}
I_2 &= \sum_{s=0}^{p}\binom{p}{s}(-1)^{m+s}\left[\frac{\mathrm{d}^{m+s}}{\mathrm{d}x^{m+s}}F(x)\right]_{x=x'}\\
&= \sum_{s=0}^{p}\binom{p}{s}(-1)^{m+s}\sum_{q=0}^{m+s}\binom{m+s}{q}\left[\frac{\mathrm{d}^{m+s-q}}{\mathrm{d}x^{m+s-q}}(x-x')^n\right]_{x=x'}\left[\frac{\mathrm{d}^{p-s+q}}{\mathrm{d}x^{p-s+q}}D_p(x)\right]_{x=x'}.
\end{aligned}
\tag{21}
$$

If $m+s-q>n$, the derivative of $(x-x')^n$ is zero; if $m+s-q<n$, the derivative of $(x-x')^n$ is proportional to $(x-x')^{n-m-s+q}$ which also vanishes for $x=x'$. Thus, the only term in the sum in (21) which is not zero is the one for which $m+s-q=n$. The integral I_2 thus is given by

$$
I_2 = \sum_{s=0}^{p}\binom{p}{s}(-1)^{m+s}\binom{m+s}{m+s-n}(n!)\left[\frac{\mathrm{d}^{p+m-n}}{\mathrm{d}x^{p+m-n}}D_p(x)\right]_{x=x'},
\tag{22}
$$

yielding

$$M_n(x',\tau) \sim (n!)\tau D_n(x') + \frac{\tau^2}{2}\sum_{p=1}^{\infty}\sum_{m=1}^{\infty}(-1)^{p+m}D_m(x')\sum_{s=0}^{p}\binom{p}{s}(-1)^{m+s}(n!)\binom{m+s}{m+s-n}\left(\frac{\mathrm{d}}{\mathrm{d}x'}\right)^{p+m-n}D_p(x')$$
$$= (n!)\tau D_n(x') + \frac{\tau^2}{2}\sum_{m=1}^{\infty}D_m(x')\left[\sum_{p=1}^{\infty}\sum_{s=0}^{p}\frac{(-1)^{p+s}\,p!(m+s)!}{s!(p-s)!(m+s-n)!}\left(\frac{\mathrm{d}}{\mathrm{d}x'}\right)^{p+m-n}D_p(x')\right].$$

The last derivative within parenthesis is of a non-negative order, i.e., $p \geq n-m$. Moreover, factorials are of non-negative integers, i.e., $s \geq n-m$. With both these conditions one arrives at a final approximation of each conditional moment as a function of the Kramers-Moyal coefficients and their derivatives, namely

$$M_n(x',\tau) \sim (n!)\tau D_n(x') + \frac{\tau^2}{2}\sum_{m=1}^{\infty}D_m(x')\sum_{p=p_0}^{\infty}\sum_{s=s_0}^{p}\frac{(-1)^{p+s}\,p!(m+s)!}{s!(p-s)!(m+s-n)!}\left(\frac{\mathrm{d}}{\mathrm{d}x'}\right)^{p+m-n}D_p(x')\,, \tag{23}$$

with $p_0 = \max(1, n-m)$ and $s_0 = \max(0, n-m)$. Equation 23 yields the Equations 13a and 13b in Gottschall and Peinke (2008) for $n=1$ and $n=2$ respectively.

Equation 23 holds a set of equations relating the conditional moments as functions of the Kramers-Moyal coefficients and their respective derivatives. In practice, one numerically computes the conditional moments and from these estimates the Kramers-Moyal coefficients. However, inverting (23) is not feasible, and a further approximation is required. Therefore, similarly to what was done for the second-order correction of the first two Kramers-Moyal coefficients (Gottschall and Peinke 2008; Rinn *et al.* 2016), we approximate (23) neglecting terms having derivatives, which implies $p+m-n=0$, i.e., $p=n-m$. Furthermore, since $p \geq \max(1, n-m)$ and $p \geq s \geq \max(0, n-m)$, one has additionally $m \leq n-1$ and $s = n-m$ respectively. Introducing these conditions in (23) yields our final approximation:

$$M_n(x',\tau) \sim (n!)\tau D_n(x') + \frac{(n!)\tau^2}{2}\sum_{m=1}^{n-1}D_m(x')D_{n-m}(x'). \tag{24}$$

Notice that the approximation of the conditional moments, as given in (24), has the practical advantage of expressing the conditional moment of order nth from the Kramers-Moyal coefficients up to order n, which enables computing recursively the Kramers-Moyal coefficients from the numerical computation of the conditional moments.

For jump processes we will need to estimate the first six Kramers-Moyal coefficients (Anvari *et al.* 2016b), which, from (24), read

$$M_1(x,\tau) = \tau D_1(x), \tag{25a}$$
$$M_2(x,\tau) = 2\tau D_2(x) + \tau^2 D_1^2(x), \tag{25b}$$
$$M_3(x,\tau) = 6\tau D_3(x) + 6\tau^2 D_1(x)D_2(x), \tag{25c}$$
$$M_4(x,\tau) = 24\tau D_4(x) + 12\tau^2\left(2D_1(x)D_3(x) + D_2^2(x)\right), \tag{25d}$$
$$M_5(x,\tau) = 120\tau D_5(x) + 120\tau^2\left(D_1(x)D_4(x) + D_2(x)D_3(x)\right), \tag{25e}$$
$$M_6(x,\tau) = 720\tau D_6(x) + 360\tau^2\left(2D_1(x)D_5(x) + 2D_2(x)D_4(x) + D_3^2(x)\right). \tag{25f}$$

Finally, inverting (25) yields (14).

The formulas for the relations (25) and the corrections F_n given by (14) are given in symbolic Python by the functions `m_formula` and `f_formula`, respectively. This numerical procedure is generalized to any maximal order N needed for the data analysis. For jump-diffusion processes $N = 6$ is sufficient.

B. Higher-order corrections and Kramers-Moyal coefficients

A more accurate approximation is possible by combining (23) and (24). More precisely, the steps to solve numerically (23) are the following ones:

- Solve (14) introducing the conditional moments, extracted directly from the data, up to order N, and derive the Kramers-Moyal coefficients $D_n(x)$ as in (13).
- Compute the derivatives up to order N_d (see discussion below).
- Introduce the derivatives of the Kramers-Moyal coefficients and the empirical conditional moments in (23) and solve it with respect to the Kramers-Moyal coefficients.
- Repeat steps S1 and S2 until the Kramers-Moyal coefficients converge within a pre-given numerical accuracy.

Moreover, since the Kramers-Moyal coefficients are typically polynomials of lower order, not larger than five or six, the derivative order N_d is a finite number, which leads to a simplification of (23). Namely, the derivative in the sum obeys $0 \leq p + m - n \leq N_d$. Thus, $p_0 \leq p \leq N_d + n - m$. Since $p_0 = \max(1, n - m)$ one has $N_d + n - m \geq 1$ and therefore the sum over m is bounded by $1 \leq m \leq N_d + n - 1$. Since $N_d > 1$, $p_0 = 1$ and therefore the sum over p is also bounded by $1 \leq p \leq N_d + n - m$.

Introducing these bounds in (23) and following steps S0-S3 above, yields the second-order approximation of the Kramers-Moyal coefficients.

Lastly, we present a general framework for obtaining all moments and Kramers-Moyal coefficients. Equation 24 can be written as

$$M_n(x', \tau) \sim (n!)\tau \hat{B}_{n,1}\left(D_1(x'), D_2(x'), \ldots, D_n(x')\right) + \frac{(n!)\tau^2}{2}\hat{B}_{n,2}\left(D_1(x'), D_2(x'), \ldots, D_{n-1}(x')\right) , \quad (26)$$

where $\hat{B}_{n,2}$ are ordinary Bell polynomials, given by

$$\hat{B}_{n,k}(x_1, x_2, \ldots, x_{n-k+1}) = \sum \frac{k!}{j_1! j_2! \cdots j_{n-k+1}!} x_1^{j_1} x_2^{j_2} \cdots x_{n-k+1}^{j_{n-k+1}} . \quad (27)$$

In the case of (26) we have $k = 2$.

The ordinary Bell's polynomials fulfill a reciprocal relation, namely any sum of ordinary Bell's polynomials of the form

$$y_n = \sum_{k=1}^{n} B_{n,k}(x_1, \ldots, x_{n-k+1}) \quad (28)$$

can be inverted as

$$x_n = \sum_{k=1}^{n} (-1)^{k-1}(k-1)!B_{n,k}(y_1, \ldots, y_{n-k+1}). \tag{29}$$

Consequently, substituting x_n and y_n by M_n and D_n, respectively, and applying (29), the inverse relation for D_n in (26) reads

$$D_n(x') = \frac{1}{\tau(n!)} \Big[\hat{B}_{n,1}\left(M_1(x',\tau), M_2(x',\tau), \ldots, M_n(x',\tau)\right) - \frac{\tau}{2}\hat{B}_{n,2}\left(M_1(x',\tau), M_2(x',\tau), \ldots, M_{n-1}(x',\tau)\right)\Big]. \tag{30}$$

This relation simply requires an iterative process for obtaining the $n-1$ conditional moments to retrieve the Kramers-Moyal coefficient D_n, which is computationally inexpensive.

This formula and its reciprocal, which accounts for the relation of the conditional moments $M_m(x,\tau)$ with the Kramers-Moyal coefficients $D_m(x)$, are implemented in **jumpdiff** in the functions `f_formula()` and `m_formula()`, respectively, where Python's symbolic language **SymPy** is used.

Affiliation:

Leonardo Rydin Gorjão
Norwegian University of Life Sciences (NMBU)
Faculty of Science and Technology
1432 Ås, Norway
and
Forschungszentrum Jülich
Institute for Energy and Climate Research – Systems Analysis and Technology Evaluation (IEK-STE)
52428 Jülich, Germany
and
Institute for Theoretical Physics
University of Cologne
50937 Köln, Germany

Dirk Witthaut
Forschungszentrum Jülich
Institute for Energy and Climate Research – Systems Analysis and Technology Evaluation (IEK-STE)
52428 Jülich, Germany
and
Institute for Theoretical Physics
University of Cologne
50937 Köln, Germany

Pedro G. Lind
Department of Computer Science
OsloMet – Oslo Metropolitan University
0130 Oslo, Norway
and
Oslo Research Center for AI
Pilestredet 52
0166 Oslo, Norway
and
NordSTAR – Nordic Center for Sustainable and Trustworthy AI Research
Pilestredet 52
0166 Oslo, Norway

Journal of Statistical Software https://www.jstatsoft.org/
published by the Foundation for Open Access Statistics https://www.foastat.org/
January 2023, Volume 105, Issue 4 *Submitted:* 2020-09-19
doi:10.18637/jss.v105.i04 *Accepted:* 2022-07-05