

[Open in app](#)[Get started](#)

Published in Towards Data Science

You have **1** free member-only story left this month. [Sign up for Medium and get an extra one](#)



Diego Barba

[Follow](#)May 3 · 7 min read ★ · [Listen](#)[Save](#)

Stochastic Processes Simulation — The Ornstein Uhlenbeck Process

Part 2 of the Stochastic Processes Simulation series. Simulating the Ornstein-Uhlenbeck process, the canonical stationary process, in Python.

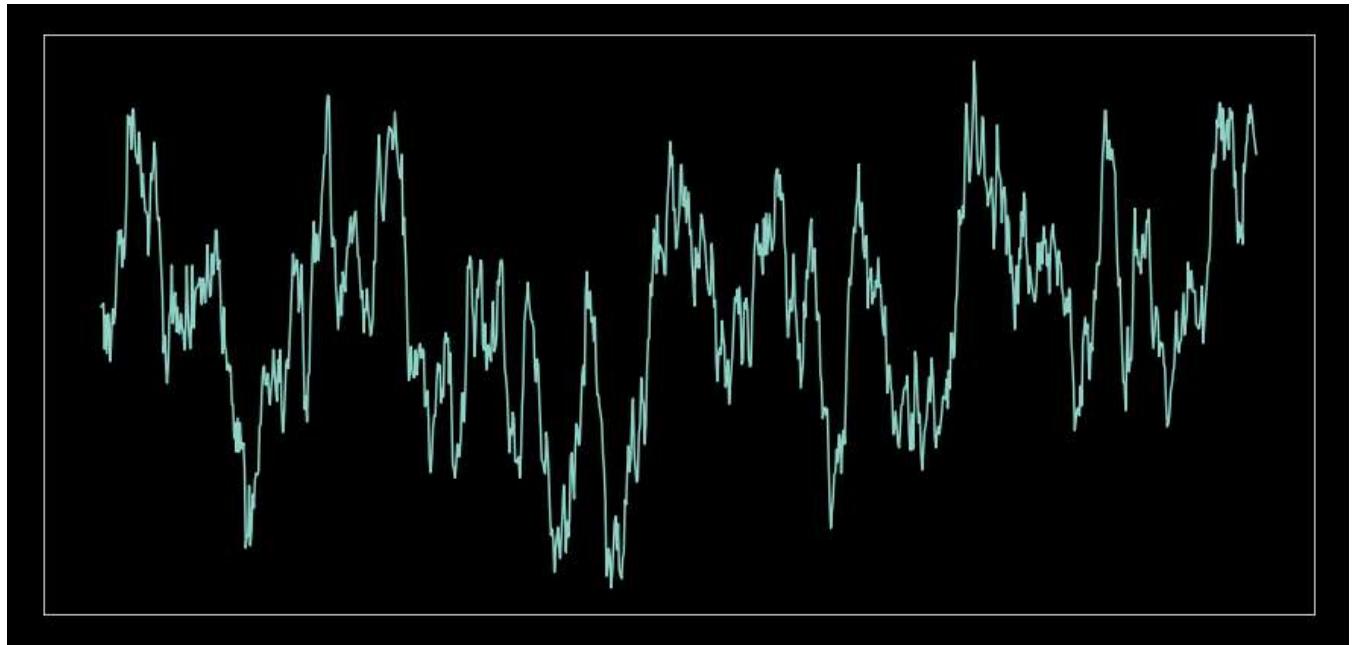


Image by author.

The quest for stationarity. Doesn't matter if we want to perform statistical inference or fit a machine learning model, we always seem to be searching for stationary features,



[Open in app](#)[Get started](#)

In this story, we will discuss the **Ornstein-Uhlenbeck process**. The canonical process among the stationary stochastic processes. **This process is not just stationary but also normally distributed.** If only time-series features behaved like this.

In the upcoming sections, we will simulate the Ornstein-Uhlenbeck process, learn how to estimate its parameters from data, and lastly, simulate multiple correlated processes. The idea is that by the end of this story you can take with you a complete neat mini-library for Ornstein-Uhlenbeck simulations.

Throughout the story, we will use tools and concepts introduced in the first story of the series about [Brownian motion](#). If you already know the basics about Brownian motion just check out the code, as we will use the Brownian motions simulation methods outlined there. Otherwise, give it a thorough read first and then come back.

Table of contents

- The equation of the process
- Simulating the process
- Estimating parameters from data
- Correlated processes
- Code summary
- Final Words

 45 | 2

The equation of the process

The Ornstein-Uhlenbeck (OU for short) process satisfies the stochastic differential equation (SDE):

$$dX_t = \alpha(\gamma - X_t)dt + \beta dW_t$$

where W_t is a [Brownian motion](#), α and β are positive constants.

The deterministic part (the drift of the process) which is the time differential term is what causes the mean reversion. When X_t is larger than γ (the asymptotic mean), the



[Open in app](#)[Get started](#)

β controls the random shocks of the process. Note that for very high values of β , the process is, asymptotically, a rescaled Brownian motion as the drift no longer has a significant effect.

There are two main reasons why this process is used as the canonical process for stationary processes:

1. The stochastic differential equation is integrable in closed-form (and we love that).
2. The process path is normally distributed. So if Brownian motion is at the center of stochastic processes because its increments are normally distributed, then the OU process is at the center of mean-reverting processes because its path is normally distributed.

The solution to the SDE is:

$$X_t = X_0 e^{-\alpha t} + \gamma(1 - e^{-\alpha t}) + \beta e^{-\alpha t} \int_0^t e^{\alpha s} dW_s$$

its asymptotic mean and variance are γ and $\beta^2 / 2\alpha$ respectively. Hence, the distribution of the process is $N(\gamma, \beta^2 / 2\alpha)$.

Simulating the process

To simulate the process we need to convert the solution of the SDE to a discrete vectorial equation, each unit time of the process is an index of a vector.

We let $t = (0, 1, 2, \dots, T-1)$, where T is the sample size. We build the arrays for the exponentials and then approximate the integral. To approximate the integral we use the cumulative sum. Note that this approximation for the integral is not suitable for very small sample sizes, so letting the sample size be greater than about 100 is a good idea.

Lets code this.

First we will define the parameters in a dataclass. Passing around many parameters

often gets messy. it's cleaner to pass them in an object of their own.



[Open in app](#)[Get started](#)

```

4  class OUParams:
5      alpha: float # mean reversion parameter
6      gamma: float # asymptotic mean
7      beta: float # Brownian motion scale (standard deviation)

```

mean_reverting_procs_1.py hosted with ❤️ by GitHub

[view raw](#)

To generate the OU process simulation we will use the code to generate Brownian motions from the [first story](#) of the series. Save the code as “brownian_motion.py”. If you don’t save it in the folder where you run the following code, you will have to change the import statement.

```

1  from typing import Optional
2
3  import numpy as np
4
5  import brownian_motion
6
7
8  def get_OU_process(
9      T: int,
10     OU_params: OUParams,
11     X_0: Optional[float] = None,
12     random_state: Optional[int] = None,
13 ) -> np.ndarray:
14     """
15         - T is the sample size.
16         - Ou_params is an instance of OUParams dataclass.
17         - X_0 the initial value for the process, if None, then X_0 is taken
18             to be gamma (the asymptotic mean).
19         Returns a 1D array.
20     """
21     t = np.arange(T, dtype=np.float128) # float to avoid np.exp overflow
22     exp_alpha_t = np.exp(-OU_params.alpha * t)
23     dW = brownian_motion.get_dW(T, random_state)
24     integral_W = _get_integral_W(t, dW, OU_params)
25     _X_0 = _select_X_0(X_0, OU_params)
26     return (
27         _X_0 * exp_alpha_t
28         + OU_params.gamma * (1 - exp_alpha_t)
29         + OU_params.beta * exp_alpha_t * integral_W

```



[Open in app](#)[Get started](#)

```
35     if X_0_in is not None:
36         return X_0_in
37     return OU_params.gamma
38
39
40 def _get_integal_W(
41     t: np.ndarray, dW: np.ndarray, OU_params: OUParams
42 ) -> np.ndarray:
43     """Integral with respect to Brownian Motion (W),  $\int \dots dW$ ."""
44     exp_alpha_s = np.exp(OU_params.alpha * t)
45     integral_W = np.cumsum(exp_alpha_s * dW)
46     return np.insert(integral_W, 0, 0)[-1]
```

Lets simulate an OU process:

```
1 OU_params = OUParams(alpha=0.07, gamma=0.0, beta=0.001)
2 OU_proc = get_OU_process(1_000, OU_params)
3
4 #-----
5 # plot
6 import matplotlib.pyplot as plt
7
8 fig = plt.figure(figsize=(15, 7))
9
10 title = "Ornstein-Uhlenbeck process, "
11 title += r"\alpha=0.07$, $\gamma = 0$, $\beta = 0.001$"
12 plt.plot(OU_proc)
13 plt.gca().set_title(title, fontsize=15)
14 plt.xticks(fontsize=15)
15 plt.yticks(fontsize=15)
```

mean_reverting_procs_3.py hosted with ❤ by GitHub

[view raw](#)

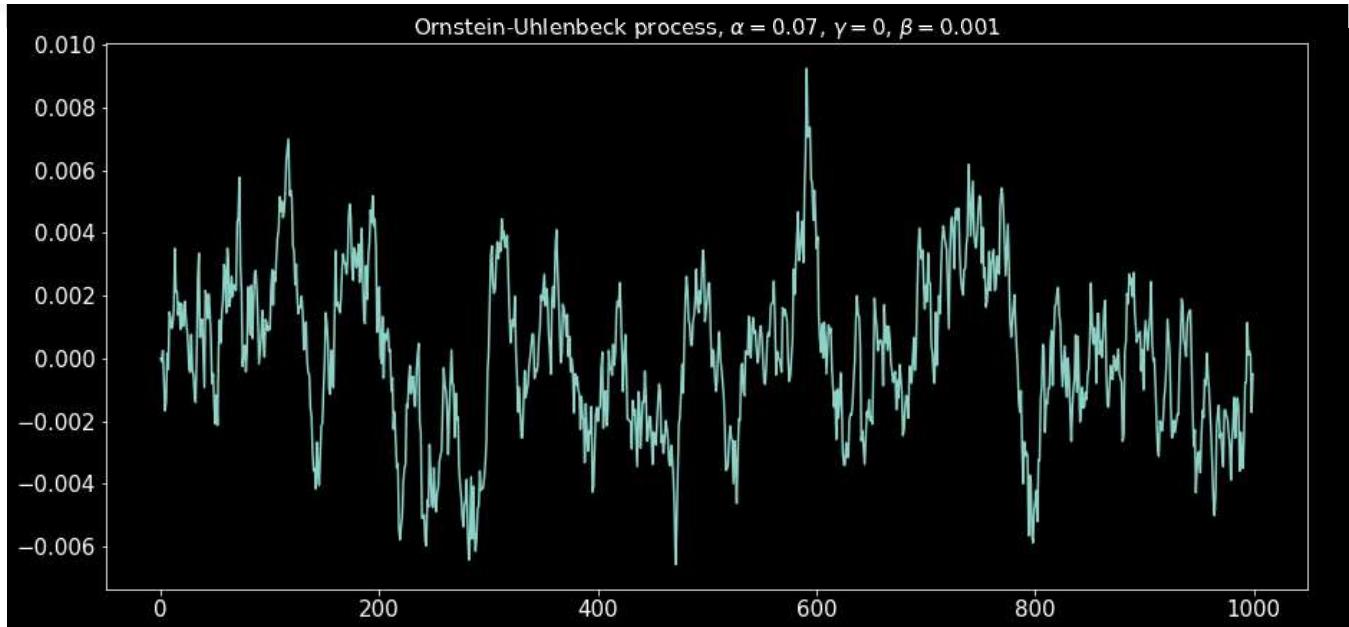

[Open in app](#)
[Get started](#)


Image by author.

Estimating parameters from data

To estimate the OU parameters from a given process we use ordinary least squares (OLS) regression.

We approximate the stochastic differential equation by a discrete equation (the time series way), known as the Euler-Maruyama method (the random difference ΔX_t is taken as a forward difference):

$$\Delta X_t = \alpha\gamma\Delta t - \alpha X_t \Delta t + \beta\epsilon_t$$

furthermore, if we make $\Delta t = 1$, then:

$$\Delta X_t = \alpha\gamma - \alpha X_t + \beta\epsilon_t$$

where ϵ is i.i.d standard normal.

Now it's easy to see the regression specification, i.e. an equation of the form $y = a + bX + \epsilon$.

The following function performs the regression and estimates the OU parameters:



[Open in app](#)[Get started](#)

```

5     ESTIMATE OU params from OLS regression.
6     - X_t is a 1D array.
7     Returns instance of OUParams.
8     """
9     y = np.diff(X_t)
10    X = X_t[:-1].reshape(-1, 1)
11    reg = LinearRegression(fit_intercept=True)
12    reg.fit(X, y)
13    # regression coefficient and constant
14    alpha = -reg.coef_[0]
15    gamma = reg.intercept_ / alpha
16    # residuals and their standard deviation
17    y_hat = reg.predict(X)
18    beta = np.std(y - y_hat)
19    return OUParams(alpha, gamma, beta)

```

mean_reverting_procs_4.py hosted with ❤ by GitHub

[view raw](#)

Let's test it with a simulated process to see if we can recover the OU params.

```

1 # generate process with random_state to reproduce results
2 OU_params = OUParams(alpha=0.07, gamma=0.0, beta=0.001)
3 OU_proc = get_OU_process(100_000, OU_params, random_state=7)
4
5 OU_params_hat = estimate_OU_params(OU_proc)

```

mean_reverting_procs_5.py hosted with ❤ by GitHub

[view raw](#)

We get:

$\alpha = 0.06845699877932063$, $\gamma = -3.9905935875610906e-05$,
 $\beta = 0.00092785693546220497295$,

the estimation is not perfect, but is good enough. To recover the true parameters we should estimate them in many samples, the averages will converge closer to the real parameters.

Correlated processes

As we saw in the [first part](#) of the series, Itô diffusive processes' random terms can be correlated. So naturally we want to have a way to create correlated OU processes. This will be useful in many models.



[Open in app](#)[Get started](#)

There are two main functionalities that we may want here:

1. All processes have the same OU parameters and they are correlated.
2. Every process has different OU parameters but processes are correlated.

We achieve this by letting the “OU_params” argument in the following code take two different types. If it is an instance of “OUParams” then it’s case 1; if it is a tuple of instances of “OUParams” then it’s case 2.

```

1  from typing import Optional, Union
2
3  import numpy as np
4
5
6  def get_corr_OU_procs(
7      T: int,
8      OU_params: Union[OUParams, tuple[OUParams, ...]],
9      n_procs: Optional[int] = None,
10     rho: Optional[float] = None,
11     random_state: Optional[int] = None,
12 ) -> np.ndarray:
13     """
14         Simulate correlated OU processes, correlation (rho) can be 0 or None.
15         - T is the sample size of the processes.
16         - OU_params can be a an instance of OUParams, in that case
17             all processes have the same parameters. It can also be a tuple,
18             in that case each process will have the parameters in the tuple,
19             each column in the resulting 2D array corresponds to the tuple index.
20         - n_procs is ignored if OU_params is tuple, else, corresponds to the number
21             of processes desired. If OU_params is not tuple and n_procs is None, will
22             raise ValueError.
23         - rho is the correlation coefficient.
24         - random_state to reproduce results.
25
26     """
27     _n_procs = _get_n_procs(OU_params, n_procs)
28     corr_dWs = brownian_motion.get_corr_dW_matrix(
29         T, _n_procs, rho, random_state
30     )
31     is_OU_params_tpl = _is_OU_params_tuple(OU_params)

```



[Open in app](#)[Get started](#)

```
37     return np.asarray(OU_procs).T
38
39
40     def _is_OU_params_tuple(
41         OU_params: Union[OUParams, tuple[OUParams, ...]]
42     ) -> bool:
43         """
44             Check if OU_params is a tuple of params,
45             return bool.
46         """
47         return isinstance(OU_params, tuple)
48
49
50     def _get_n_procs(
51         OU_params: Union[OUParams, tuple[OUParams, ...]], n_procs: Optional[int]
52     ) -> int:
53         """
54             Define the number of processes, if Ou_params is a tuple the
55             number of processes is the lenght of the tuple. If it is not a tuple
56             then it is the "n_procs" supplied as argument,
57             if it is None will raise ValueError.
58         """
59         if _is_OU_params_tuple(OU_params):
60             return len(OU_params) # type: ignore
61         elif n_procs is None:
62             raise ValueError("If OU_params is not tuple, n_procs cannot be None.")
63         return n_procs
64
65
66     def _get_OU_params_i(
67         OU_params: Union[OUParams, tuple[OUParams, ...]],
68         i: int,
69         is_OU_params_tpl: bool,
70     ) -> OUParams:
71         """
72             Returns the ith value of the OU_params tuple if it is a tuple,
73             otherwise returns OUParams.
74         """
75         if is_OU_params_tpl:
76             return OU_params[i] # type: ignore
77         return OU_params # type: ignore
78
```





Open in app

Get started

```

84     SIMULATES THE OU PROCESS WITH AN EXTERNAL W.
85     X_0 IS TAKEN AS THE ASYMPTOTIC MEAN GAMMA FOR SIMPLICITY.
86     """
87     t = np.arange(T, dtype=np.float128) # float to avoid np.exp overflow
88     exp_alpha_t = np.exp(-OU_params.alpha * t)
89     integral_W = _get_integal_W(t, dW, OU_params)
90     return (
91         OU_params.gamma * exp_alpha_t
92         + OU_params.gamma * (1 - exp_alpha_t)
93         + OU_params.hata * exp_alpha_t * integral_W

```

```

1  # case 1
2
3  T = 1_000
4  OU_params = OUParams(alpha=0.07, gamma=0.0, beta=0.001)
5  n_proc = 5
6  rho = 0.9
7  OU_procs = get_corr_OU_procs(T, OU_params, n_proc, rho)
8
9  #-----
10 # plot
11 import matplotlib.pyplot as plt
12 import seaborn as sns
13
14 fig = plt.figure(figsize=(15, 5))
15
16 title = "Correlated Ornstein-Uhlenbeck processes, single params"
17 plt.subplot(1, 2, 1)
18 plt.plot(OU_procs)
19 plt.gca().set_title(title, fontsize=15)
20 plt.xticks(fontsize=15)
21 plt.yticks(fontsize=15)
22
23 title = "Correlation matrix (increments) heatmap"
24 plt.subplot(1, 2, 2)
25 sns.heatmap(np.corrcoef(np.diff(OU_procs, axis=0), rowvar=False), cmap="mako")
26 plt.gca().set_title(title, fontsize=15)
27 plt.xticks(fontsize=15)
28 plt.yticks(fontsize=15)

```





Open in app

Get started

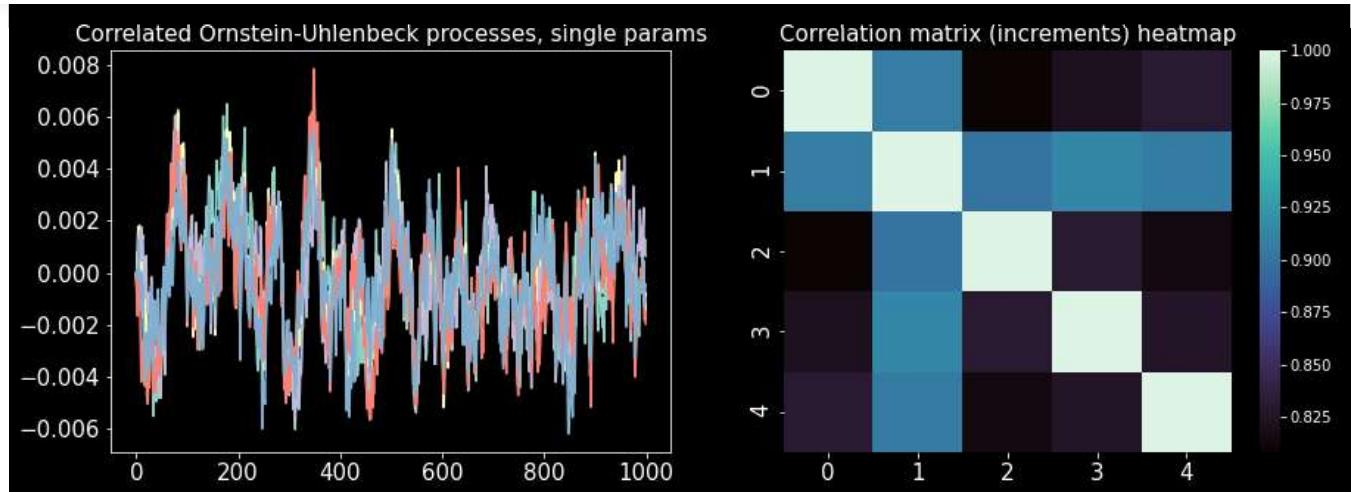


Image by author.

```

1 # case 2
2
3 T = 1_000
4 OU_params = (
5     OUParams(alpha=0.07, gamma=0.0, beta=0.005),
6     OUParams(alpha=0.05, gamma=0.0, beta=0.003),
7     OUParams(alpha=0.06, gamma=0.0, beta=0.002),
8     OUParams(alpha=0.09, gamma=0.0, beta=0.002),
9     OUParams(alpha=0.08, gamma=0.0, beta=0.001),
10 )
11 rho = 0.9
12 OU_procs = get_corr_OU_procs(T, OU_params, n_proc, rho)
13
14 #-----
15 # plot
16 import matplotlib.pyplot as plt
17 import seaborn as sns
18
19 fig = plt.figure(figsize=(15, 5))
20
21 title = "Correlated Ornstein-Uhlenbeck processes, multi params"
22 plt.subplot(1, 2, 1)
23 plt.plot(OU_procs)
24 plt.gca().set_title(title, fontsize=15)
25 plt.xticks(fontsize=15)
26 plt.yticks(fontsize=15)
27
28 title = "Correlation matrix (increments) heatmap"

```





Open in app

Get started

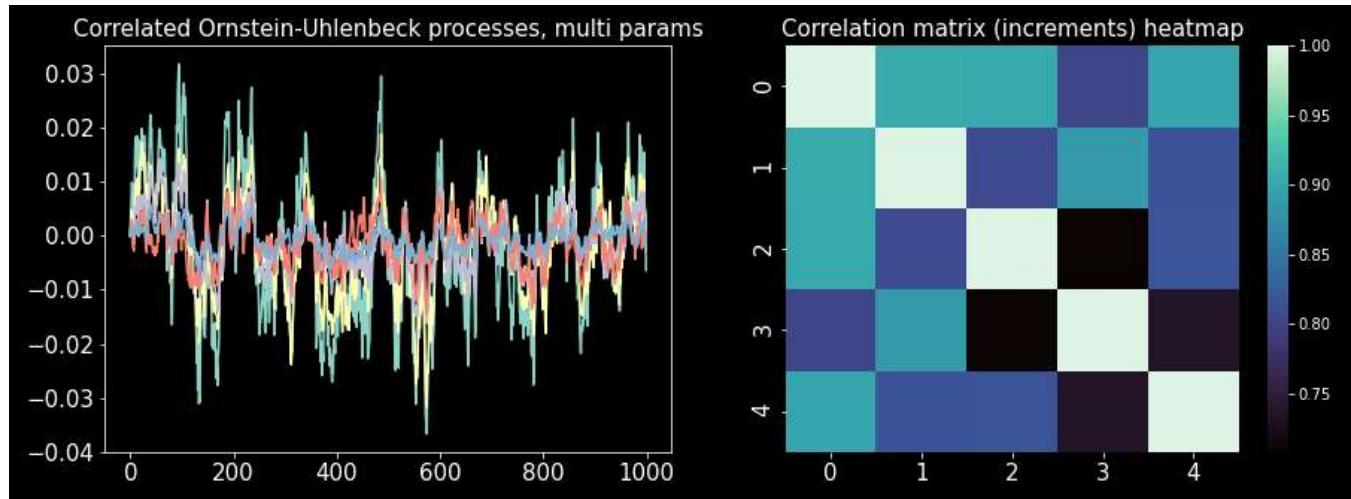


Image by author.

Code summary

As usual, I like to wrap up the story with a summary of all the code used throughout the sections and produce a single coherent mini-library that we can save for later use.

```

1  """OU_proc.py"""
2
3  from dataclasses import dataclass
4  from typing import Optional, Union
5
6  import numpy as np
7
8  from sklearn.linear_model import LinearRegression
9
10 import brownian_motion
11
12 # -----
13 # OU process
14
15
16 @dataclass
17 class OUParams:
18     alpha: float # mean reversion parameter
19     gamma: float # asymptotic mean
20     beta: float # Brownian motion scale (standard deviation)
21

```



[Open in app](#)[Get started](#)

```
27     random_state: Optional[int] = None,
28 ) -> np.ndarray:
29     """
30     - T is the sample size.
31     - Ou_params is an instance of OUParams dataclass.
32     - X_0 the initial value for the process, if None, then X_0 is taken
33         to be gamma (the asymptotic mean).
34     Returns a 1D array.
35     """
36     t = np.arange(T, dtype=np.float128) # float to avoid np.exp overflow
37     exp_alpha_t = np.exp(-OU_params.alpha * t)
38     dW = brownian_motion.get_dW(T, random_state)
39     integral_W = _get_integal_W(t, dW, OU_params)
40     _X_0 = _select_X_0(X_0, OU_params)
41     return (
42         _X_0 * exp_alpha_t
43         + OU_params.gamma * (1 - exp_alpha_t)
44         + OU_params.beta * exp_alpha_t * integral_W
45     )
46
47
48 def _select_X_0(X_0_in: Optional[float], OU_params: OUParams) -> float:
49     """Returns X_0 input if not none, else gamma (the long term mean)."""
50     if X_0_in is not None:
51         return X_0_in
52     return OU_params.gamma
53
54
55 def _get_integal_W(
56     t: np.ndarray, dW: np.ndarray, OU_params: OUParams
57 ) -> np.ndarray:
58     """Integral with respect to Brownian Motion (W), ∫...dW."""
59     exp_alpha_s = np.exp(OU_params.alpha * t)
60     integral_W = np.cumsum(exp_alpha_s * dW)
61     return np.insert(integral_W, 0, 0)[-1]
62
63
64 # -----
65 # OU parameters from data with OLS.
66
67
68 def estimate_OU_params(X_t: np.ndarray) -> OUParams:
69     """
```



[Open in app](#)[Get started](#)

```
74     y = np.array(y)
75     X = X_t[:-1].reshape(-1, 1)
76     reg = LinearRegression(fit_intercept=True)
77     reg.fit(X, y)
78     # regression coefficient and constant
79     alpha = -reg.coef_[0]
80     gamma = reg.intercept_ / alpha
81     # residuals and their standard deviation
82     y_hat = reg.predict(X)
83     beta = np.std(y - y_hat)
84     return OUParams(alpha, gamma, beta)
85
86
87     # -----
88     # Correlated OU processes
89
90
91     def get_corr_OU_procs(
92         T: int,
93         OU_params: Union[OUParams, tuple[OUParams, ...]],
94         n_procs: Optional[int] = None,
95         rho: Optional[float] = None,
96         random_state: Optional[int] = None,
97     ) -> np.ndarray:
98         """
99             Simulate correlated OU processes, correlation (rho) can be 0 or None.
100            - T is the sample size of the processes.
101            - OU_params can be a an instance of OUParams, in that case
102                all processes have the same parameters. It can also be a tuple,
103                in that case each process will have the parameters in the tuple,
104                each column in the resulting 2D array corresponds to the tuple index.
105            - n_procs is ignored if OU_params is tuple, else, corresponds to the number
106                of processes desired. If OU_params is not tuple and n_procs is None, will
107                raise ValueError.
108            - rho is the correlation coefficient.
109            - random_state to reproduce results.
110
111        """
112        _n_procs = _get_n_procs(OU_params, n_procs)
113        corr_dWs = brownian_motion.get_corr_dW_matrix(
114            T, _n_procs, rho, random_state
115        )
116        is_OU_params_tpl = _is_OU_params_tuple(OU_params)
```



[Open in app](#)[Get started](#)

```
122     return np.asarray(OU_procs).T
123
124
125     def _is_OU_params_tuple(
126         OU_params: Union[OUParams, tuple[OUParams, ...]]
127     ) -> bool:
128         """
129             Check if OU_params is a tuple of params,
130             return bool.
131         """
132         return isinstance(OU_params, tuple)
133
134
135     def _get_n_procs(
136         OU_params: Union[OUParams, tuple[OUParams, ...]], n_procs: Optional[int]
137     ) -> int:
138         """
139             Define the number of processes, if OU_params is a tuple the
140             number of processes is the lenght of the tuple. If it is not a tuple
141             then it is the "n_procs" supplied as argument,
142             if it is None will raise ValueError.
143         """
144         if _is_OU_params_tuple(OU_params):
145             return len(OU_params) # type: ignore
146         elif n_procs is None:
147             raise ValueError("If OU_params is not tuple, n_procs cannot be None.")
148         return n_procs
149
150
151     def _get_OU_params_i(
152         OU_params: Union[OUParams, tuple[OUParams, ...]],
153         i: int,
154         is_OU_params_tpl: bool,
155     ) -> OUParams:
156         """
157             Returns the ith value of the OU_params tuple if it is a tuple,
158             otherwise returns OUParams.
159         """
160         if is_OU_params_tpl:
161             return OU_params[i] # type: ignore
162         return OU_params # type: ignore
163
164
```



[Open in app](#)[Get started](#)

```
165 SIMULATES THE OU PROCESS WITH AN EXTERNAL dw.
166
167 X_0 is taken as the asymptotic mean gamma for simplicity.
168 """
169
170 t = np.arange(T, dtype=np.float128) # float to avoid np.exp overflow
171 exp_alpha_t = np.exp(-OU_params.alpha * t)
172 integral_W = _get_integral_W(t, dw, OU_params)
173
174 return (
```

and estimate its parameters from data. This is extremely useful for Monte Carlo simulations, once we know the parameters of a process we want to study then we can generate many Monte Carlo trials to get some statistics and further analysis.

Stay tuned for the next story in the series.

References

- [1] B. Øksendal, [Stochastic Differential Equations](#) (2013), Sixth Edition, Springer.

Stay connected for the next stories of the series. Follow me on [Medium](#) and subscribe to get the updates of the next stories as soon as they come out.

Get an email whenever Diego Barba publishes.

Get an email whenever Diego Barba publishes. By signing up, you will create a Medium account if you don't already have...

[medium.com](https://medium.com/@diego_barba)

I hope this story was useful to you. If I missed anything, please let me know.

Liked the story? Become a Medium member through my referral link and get unlimited access to my stories and many others.



[Open in app](#)[Get started](#)

writers you read, and you get full access to every story...

[medium.com](https://medium.com/@diegobarba/stochastic-processes-simulation-the-ornstein-uhlenbeck-process-e8bfff820f3)

Navigate through the Stochastic Processes Simulation series

Previous story in the series:

Stochastic Processes Simulation — Brownian Motion, The Basics

Part 1 of the Stochastic Processes Simulation series. Simulate correlated Brownian motions in Python from scratch.

[towardsdatascience.com](https://towardsdatascience.com/stochastic-processes-simulation-brownian-motion-the-basics-1-1e8bfff820f3)

Next stories in the series:

Stochastic Processes Simulation — The Cox-Ingersoll-Ross Process

Part 3 of the Stochastic Processes Simulation series. Simulating the Cox-Ingersoll-Ross process in Python from scratch.

[towardsdatascience.com](https://towardsdatascience.com/stochastic-processes-simulation-the-cox-ingersoll-ross-process-1-1e8bfff820f3)

Stochastic Processes Simulation — Geometric Brownian Motion

Part 4 of the Stochastic Processes Simulation series. Simulating geometric Brownian motion in Python from scratch.

[towardsdatascience.com](https://towardsdatascience.com/stochastic-processes-simulation-geometric-brownian-motion-1-1e8bfff820f3)

Stochastic Processes Simulation — Generalized Geometric Brownian Motion



[Open in app](#)[Get started](#)

Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. [Take a look.](#)

[Get this newsletter](#)

[About](#) [Help](#) [Terms](#) [Privacy](#)

Get the Medium app

