stochastic Documentation

Release 0.7.0

Christopher Flynn

Jul 12, 2022

CONTENTS

Stochastic is a python package for generating realizations of stochastic processes.

CHAPTER

ONE

INSTALLATION

Stochastic is available on pypi and can be installed using `pip`:

```
pip install stochastic
```

CHAPTER

TWO

DEPENDENCIES

Stochastic depends on `numpy` for most calculations and `scipy` for certain random variable generation.

CHAPTER

THREE

COMPATIBILITY

Stochastic is tested on Python versions 3.6, 3.7, and 3.8.

CHAPTER

FOUR

PERFORMANCE

This package uses `numpy` and `scipy` wherever possible for faster computation. For improved performance under Monte Carlo simulation, some classes will store results of intermediate computations for faster generation on subsequent simulations.

CHAPTER

FIVE

DOCUMENTATION

5.1 General Usage

5.1.1 Processes

This package offers a number of common discrete-time, continuous-time, and noise process objects for generating realizations of stochastic processes as `numpy` arrays.

The diffusion processes are approximated using the Euler–Maruyama method.

Here are the currently supported processes and how to access their classes:

- stochastic.processes
  - continuous
    - BesselProcess
    - BrownianBridge
    - BrownianExcursion
    - BrownianMeander
    - BrownianMotion
    - CauchyProcess
    - FractionalBrownianMotion
    - GammaProcess
    - GeometricBrownianMotion
    - InverseGaussianProcess
    - MixedPoissonProcess
    - MultifractionalBrownianMotion
    - PoissonProcess
    - SquaredBesselProcess
    - VarianceGammaProcess
    - WienerProcess
  - diffusion
    - DiffusionProcess (generalized)

    * ConstantElasticityVarianceProcess
    * CoxIngersollRossProcess
    * ExtendedVasicekProcess
    * OrnsteinUhlenbeckProcess
    * VasicekProcess
  - discrete
    * BernoulliProcess
    * ChineseRestaurantProcess
    * DirichletProcess
    * MarkovChain
    * MoranProcess
    * RandomWalk
  - noise
    * BlueNoise
    * BrownianNoise
    * ColoredNoise
    * PinkNoise
    * RedNoise
    * VioletNoise
    * WhiteNoise
    * FractionalGaussianNoise
    * GaussianNoise

5.1.2 Usage patterns

The sample() method

To use `stochastic`, import the process you want and instantiate with the required parameters. Every process class has a `sample` method for generating realizations. The `sample` methods accept a parameter `n` for the quantity of steps in the realization, but others (Poisson, for instance) may take additional parameters. Parameters can be accessed as attributes of the instance.

```
from stochastic.processes.discrete import BernoulliProcess


bp = BernoulliProcess(p=0.6)
s = bp.sample(16)
success_probability = bp.p
```

Continuous processes provide a default parameter, `t`, which indicates the maximum time of the process realizations. The default value is 1. The sample method will generate `n` equally spaced increments on the interval `[0, t]`.

The sample_at() method

Some continuous processes also provide a `sample_at()` method, in which a sequence of time values can be passed at which the object will generate a realization. This method ignores the parameter, `t`, specified on instantiation.

```
from stochastic.processes.continuous import BrownianMotion


bm = BrownianMotion(drift=1, scale=1, t=1)
times = [0, 3, 10, 11, 11.2, 20]
s = sample_at(times)
```

The times() method

Continuous-time processes also provide a method `times()` which generates the time values (using `numpy.linspace`) corresponding to a realization of `n` steps. This is particularly useful for plotting your samples.

```
import matplotlib.pyplot as plt
from stochastic.processes.continuous import FractionalBrownianMotion


fbm = FractionalBrownianMotion(hurst=0.7, t=1)
s = fbm.sample(32)
times = fbm.times(32)

plt.plot(times, s)
plt.show()
```

The algorithm option

Some processes provide an optional parameter `algorithm`, in which one can specify which algorithm to use to generate the realization using the `sample()` or `sample_at()` methods. See class-specific documentation for implementations.

```
from stochastic.processes.noise import FractionalGaussianNoise


fgn = FractionalGaussianNoise(hurst=0.6, t=1)
s = fgn.sample(32, algorithm='hosking')
```

5.2 Random Number Generation

5.2.1 Numpy's random number generation

Stochastic relies on numpy for random number generation. Since numpy 1.17, the newer `Generator` objects provide improved performance:

Note: From numpy docs: The Generator's normal, exponential and gamma functions use 256-step Ziggurat methods which are 2-10 times faster than NumPy's Box-Muller or inverse CDF implementations.

By default, the stochastic package uses numpy's faster Generator random number generation. With a function call, we can change the default back to the legacy random number generation, which uses RandomState objects.

If no `rng` arg is passed when instantiating process instances, each instance will reference the `stochastic.random` module's `generator` attribute for random number generation.

5.2.2 Examples

Changing the default random number generation on instances without specified `rng`:

```
from stochastic.processes import GaussianNoise
from stochastic import random

gn = GaussianNoise()
print(gn.rng)
# Generator(PCG64)

# use the legacy random number generator
random.use_randomstate()

print(gn.rng)
# <module 'numpy.random' from '/path/to/site-packages/numpy/random/__init__.py'>

# use the newer Generator
random.use_generator()

print(gn.rng)
# Generator(PCG64)
```

Setting the seed value:

```
from stochastic.processes import GaussianNoise
from stochastic import random

gn = GaussianNoise()
print(gn.rng)
# Generator(PCG64)

random.seed(42)
print(gn.rng.bit_generator.state)
# {'bit_generator': 'PCG64', 'state': {'state': 274674114334540486603088602300644985544, 'inc':␣
↪332724090758049132448979897138935081983}, 'has_uint32': 0, 'uinteger': 0}
print(gn.sample(4))
# [ 0.15235854 -0.51999205  0.3752256   0.47028236]

random.seed(42)
print(gn.rng.bit_generator.state)
# {'bit_generator': 'PCG64', 'state': {'state': 274674114334540486603088602300644985544, 'inc':␣
↪332724090758049132448979897138935081983}, 'has_uint32': 0, 'uinteger': 0}
print(gn.sample(4))
# [ 0.15235854 -0.51999205  0.3752256   0.47028236]
```

Passing custom generators to process instances at instantiation:

```python
from numpy.random import Generator
from numpy.random import PCG64
from stochastic.processes import GaussianNoise
from stochastic import random

generator = Generator(PCG64(seed=42))

gn = GaussianNoise(rng=generator)
# generator specific to this gaussian noise instance
print(gn.rng.bit_generator.state)
# {'bit_generator': 'PCG64', 'state': {'state': 274674114334540486603088602300644985544, 'inc':
↪332724090758049132448979897138935081983}, 'has_uint32': 0, 'uinteger': 0}

# stochastic's global generator, different from the one attached to `gn`
print(random.generator.bit_generator.state)
# {'bit_generator': 'PCG64', 'state': {'state': 228239801863081385502825691348763076514, 'inc':
↪61631449755775032062670113901777656135}, 'has_uint32': 0, 'uinteger': 0}
```

5.2.3 Documentation

`stochastic.random.`**`generator = Generator(PCG64) at 0x3FFB49E4F20`**

The default random number generator for the stochastic package

`stochastic.random.`**`seed`**`(`*value*`)`

Sets the seed for numpy legacy or `default_rng` generators.

If using the legacy generator, this will call `numpy.random.seed(value)`. Otherwise a new random number generator is created using `numpy.random.default_rng(value)`.

`stochastic.random.`**`use_generator`**`(`*rng=None*`)`

Use the new numpy Generator as default for stochastic.

Sets the default random number generator for stochastic processes to the newer `np.random.default_rng()`.

Note: This is the default generator and there is no need to call this function unless returning to the default after switching away from it.

Parameters **rng** (*numpy.random.Generator*) – a Generator instance to use as the default random number generator for stochastic.

`stochastic.random.`**`use_randomstate`**`(`*rng=None*`)`

Use the legacy numpy RandomState generator as default for stochastic.

Sets the default random number generator for stochastic processes to the legacy `np.random`.

Parameters **rng** (*numpy.random.RandomState*) – a RandomState instance to use as the default random number generator for stochastic.

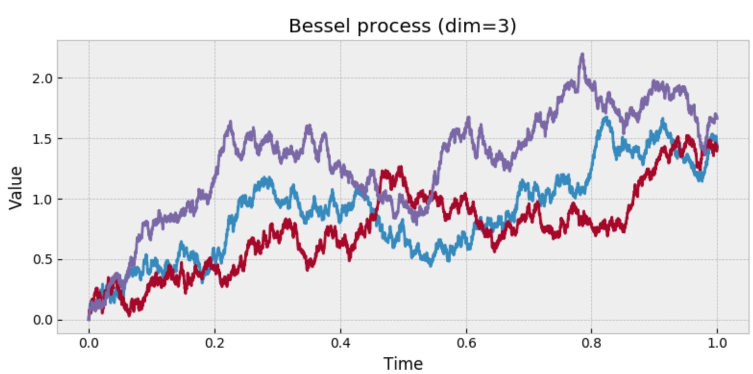
5.3 Continuous-time Processes

The `stochastic.processes.continuous` module provides classes for generating discretely sampled continuous-time stochastic processes.

- *stochastic.processes.continuous.BesselProcess*
- *stochastic.processes.continuous.BrownianBridge*
- *stochastic.processes.continuous.BrownianExcursion*
- *stochastic.processes.continuous.BrownianMeander*
- *stochastic.processes.continuous.BrownianMotion*
- *stochastic.processes.continuous.CauchyProcess*
- *stochastic.processes.continuous.FractionalBrownianMotion*
- *stochastic.processes.continuous.GammaProcess*
- *stochastic.processes.continuous.GeometricBrownianMotion*
- *stochastic.processes.continuous.InverseGaussianProcess*
- *stochastic.processes.continuous.MixedPoissonProcess*
- *stochastic.processes.continuous.MultifractionalBrownianMotion*
- *stochastic.processes.continuous.PoissonProcess*
- *stochastic.processes.continuous.SquaredBesselProcess*
- *stochastic.processes.continuous.VarianceGammaProcess*
- *stochastic.processes.continuous.WienerProcess*

class stochastic.processes.continuous.**BesselProcess**(*dim=1*, *t=1*, *rng=None*)

Bessel process.



The Bessel process is the Euclidean norm of an n-dimensional Wiener process, e.g. $\|\mathbf{W}_t\|$

Generate Bessel process realizations using `dim` independent Brownian motion processes on the interval $[0, t]$

Parameters

- **dim** (*int*) – the number of underlying independent Brownian motions to use
- **t** (*float*) – the right hand endpoint of the time interval $[0, t]$ for the process
- **rng** (*numpy.random.Generator*) – a custom random number generator

sample(*n*)
Generate a realization.

Parameters **n** (*int*) – the number of increments to generate

sample_at(*times*)
Generate a realization using specified times.

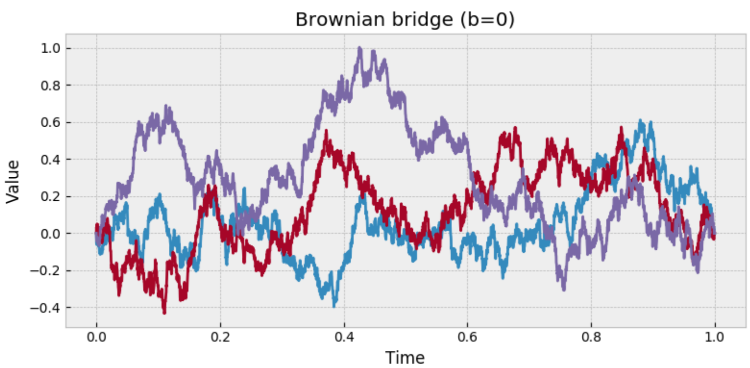
Parameters **times** – a vector of increasing time values at which to generate the realization

property t
End time of the process.

times(*n*)
Generate times associated with n increments on [0, t].

Parameters **n** (*int*) – the number of increments

class stochastic.processes.continuous.**BrownianBridge**(*b=0*, *t=1*, *rng=None*)
Brownian bridge.



A Brownian bridge is a Brownian motion with a conditional value on the right endpoint of the process.

Parameters

- **b** (*float*) – the right endpoint value of the Brownian bridge at time t
- **t** (*float*) – the right hand endpoint of the time interval $[0, t]$ for the process
- **rng** (*numpy.random.Generator*) – a custom random number generator

property b
Right endpoint value.

sample(*n*)
Generate a realization.

Parameters **n** (*int*) – the number of increments to generate

sample_at(*times*, *b=None*)
Generate a realization using specified times.

Parameters

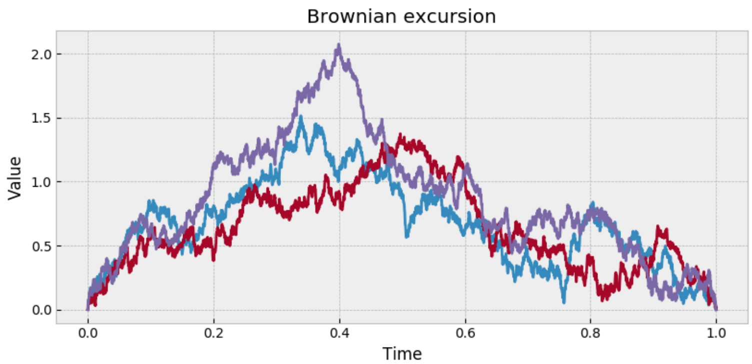
- **times** – a vector of increasing time values at which to generate the realization
- **b** (*float*) – the right endpoint value for `times` [-1]

property t
End time of the process.

times(*n*)
Generate times associated with n increments on [0, t].

Parameters **n** (*int*) – the number of increments

class stochastic.processes.continuous.**BrownianExcursion**(*t=1*, *rng=None*)
Brownian excursion.



A Brownian excursion is a Brownian bridge from (0, 0) to (t, 0) which is conditioned to be nonnegative on the interval [0, t].

Generated using method by

- Biane, Philippe. "Relations entre pont et excursion du mouvement Brownien reel." Ann. Inst. Henri Poincare 22, no. 1 (1986): 1-7.
- Vervaat, Wim. "A relation between Brownian bridge and Brownian excursion." The Annals of Probability (1979): 143-149.

Parameters

- **t** (*float*) – the right hand endpoint of the time interval $[0, t]$ for the process
- **rng** (*numpy.random.Generator*) – a custom random number generator

sample(*n*)
Generate a realization.

Parameters **n** (*int*) – the number of increments to generate.

sample_at(*times*)
Generate a realization using specified times.

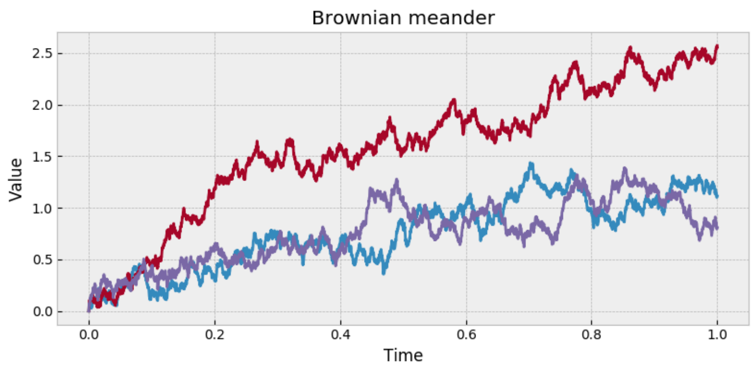
Parameters **times** – a vector of increasing time values at which to generate the realization

property t
End time of the process.

times(*n*)
Generate times associated with n increments on [0, t].

Parameters **n** (*int*) – the number of increments

class stochastic.processes.continuous.**BrownianMeander**(*t=1*, *rng=None*)
Brownian meander process.



A Brownian motion conditioned such that the process is nonnegative.

Generated using method by

- Williams, David. "Decomposing the Brownian path." Bulletin of the American Mathematical Society 76, no. 4 (1970): 871-873.
- Imhof, J-P. "Density factorizations for Brownian motion, meander and the three-dimensional Bessel process, and applications." Journal of Applied Probability 21, no. 3 (1984): 500-510.

Parameters

- **t** (*float*) – the right hand endpoint of the time interval $[0, t]$ for the process
- **rng** (*numpy.random.Generator*) – a custom random number generator

sample(*n*, *b=None*)
Generate a realization.

Parameters

- **n** (*int*) – the number of increments to generate
- **b** (*float*) – the nonnegative right hand endpoint of the meander. If not provided, one is randomly selected from a $\sqrt{2E}$ random variable where E is exponential.

sample_at(*times*, *b=None*)
Generate a realization using specified times.

Parameters

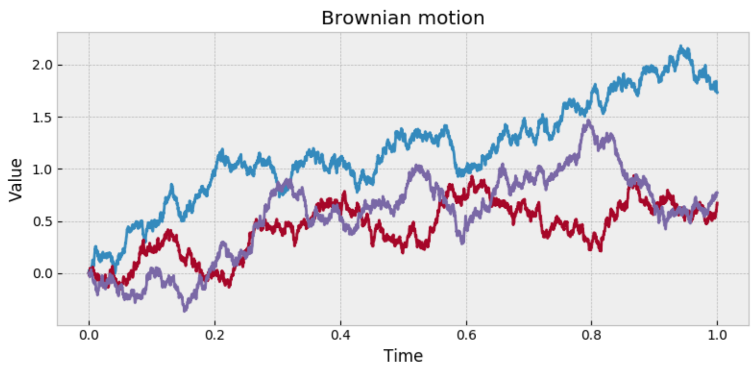
- **times** – a vector of increasing time values at which to generate the realization
- **b** (*float*) – the right endpoint value for `times` [-1]. If not provided, one is randomly selected from a $\sqrt{2tE}$ random variable where E is exponential and t is `times` [-1].

property t
End time of the process.

times(*n*)
Generate times associated with n increments on [0, t].

Parameters **n** (*int*) – the number of increments

class `stochastic.processes.continuous.`**BrownianMotion**(*drift=0*, *scale=1*, *t=1*, *rng=None*)
Brownian motion.



A standard Brownian motion (discretely sampled) has independent and identically distributed Gaussian increments with variance equal to increment length. Non-standard Brownian motion includes a linear drift parameter and scale factor.

Parameters

- **drift** (*float*) – rate of change of the expected value
- **scale** (*float*) – scale factor of the Gaussian process
- **t** (*float*) – the right hand endpoint of the time interval $[0, t]$ for the process
- **rng** (*numpy.random.Generator*) – a custom random number generator

property drift
Drift parameter.

sample(*n*)
Generate a realization.

Parameters **n** (*int*) – the number of increments to generate

sample_at(*times*)
Generate a realization using specified times.

Parameters **times** – a vector of increasing time values at which to generate the realization

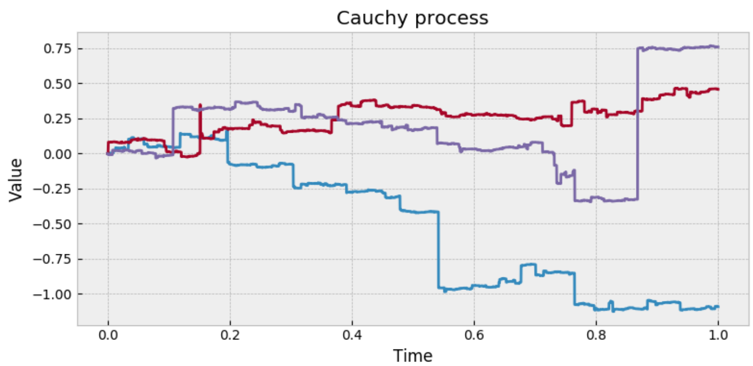
property scale
Scale parameter.

property t
End time of the process.

times(*n*)
Generate times associated with n increments on [0, t].

Parameters **n** (*int*) – the number of increments

class stochastic.processes.continuous.**CauchyProcess**(*t=1*, *rng=None*)
Symmetric Cauchy process.



The symmetric Cauchy process is a Brownian motion with a Levy subordinator using location parameter 0 and scale parameter $t^2/2$.

Parameters

- **t** (*float*) – the right hand endpoint of the time interval $[0, t]$ for the process
- **rng** (*numpy.random.Generator*) – a custom random number generator

sample(*n*)

Generate a realization.

Parameters **n** (*int*) – the number of increments to generate.

sample_at(*times*)

Generate a realization using specified times.

Parameters **times** – a vector of increasing time values at which to generate the realization

property t

End time of the process.

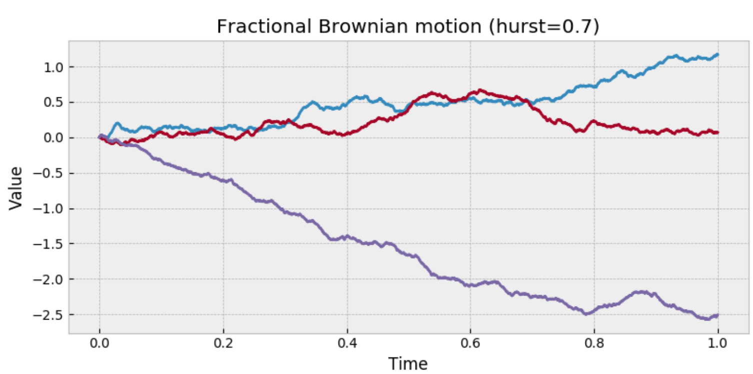
times(*n*)

Generate times associated with n increments on [0, t].

Parameters **n** (*int*) – the number of increments

class stochastic.processes.continuous.**FractionalBrownianMotion**(*hurst=0.5*, *t=1*, *rng=None*)

Fractional Brownian motion process.



A fractional Brownian motion (discretely sampled) has correlated Gaussian increments defined by Hurst parameter H. When $H = 1/2$, the process is a standard Brownian motion. When $H > 1/2$, the increments are positively correlated. When $H < 1/2$, the increments are negatively correlated.

Hosking's method:

- Hosking, Jonathan RM. "Modeling persistence in hydrological time series using fractional differencing." Water resources research 20, no. 12 (1984): 1898-1908.

Davies Harte method:

- Davies, Robert B., and D. S. Harte. "Tests for Hurst effect." Biometrika 74, no. 1 (1987): 95-101.

Parameters

- **hurst** (*float*) – the Hurst parameter on the interval (0, 1)
- **t** (*float*) – the right hand endpoint of the time interval $[0, t]$ for the process
- **rng** (*numpy.random.Generator*) – a custom random number generator

property hurst

Hurst parameter.

sample(*n*)

Generate a realization.

Parameters **n** (*int*) – the number of increments to generate

property t

End time of the process.

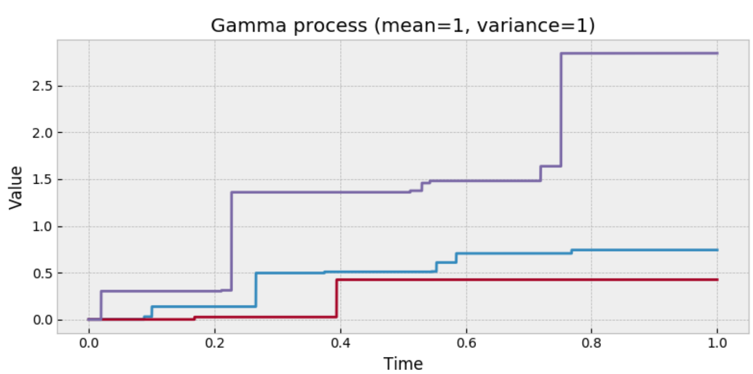
times(*n*)

Generate times associated with n increments on [0, t].

Parameters **n** (*int*) – the number of increments

class stochastic.processes.continuous.**GammaProcess**(*mean=None, variance=None, rate=None, scale=None, t=1, rng=None*)

Gamma process.



A Gamma process (discretely sampled) is the summation of stationary independent increments which are distributed as gamma random variables. This class supports instantiation using the mean/variance parametrization or the rate/scale parametrization.

Parameters

- **mean** (*float*) – mean increase per unit time; supply with `variance`
- **variance** (*float*) – variance of increase per unit time; supply with `mean`
- **rate** (*float*) – the rate of jump arrivals; supply with `scale`
- **scale** (*float*) – the size of the jumps; supple with `rate`
- **t** (*float*) – the right hand endpoint of the time interval $[0, t]$ for the process

- **rng** (*numpy.random.Generator*) – a custom random number generator

property mean
Mean increase per unit time.

property rate
Rate of jump arrivals.

sample(*n*)
Generate a realization.

Parameters **n** (*int*) – the number of increments to generate

sample_at(*times*)
Generate a realization at specified times.

Parameters **times** – a vector of increasing time values at which to generate the realization

property scale
Scale parameter for jump sizes.

property t
End time of the process.

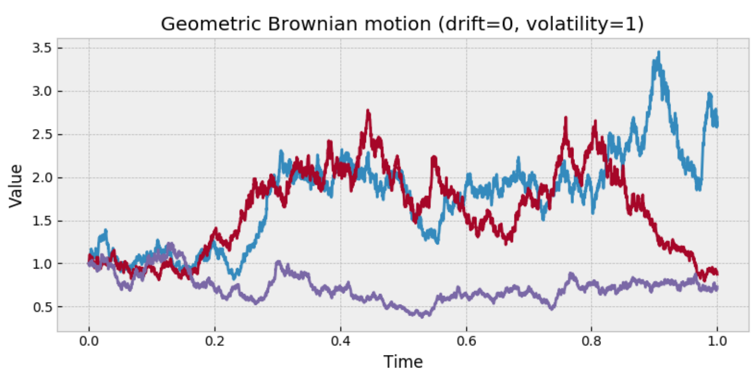
times(*n*)
Generate times associated with n increments on [0, t].

Parameters **n** (*int*) – the number of increments

property variance
Variance of increase per unit time.

class stochastic.processes.continuous.**GeometricBrownianMotion**(*drift=0*, *volatility=1*, *t=1*, *rng=None*)

Geometric Brownian motion process.



A geometric Brownian motion S_t is the analytic solution to the stochastic differential equation with Wiener process W_t:

$$dS_t = \mu S_t dt + \sigma S_t dW_t$$

and can be represented with initial value S_0 in the form:

$$S_t = S_0 \exp \left(\left(\mu - \frac{\sigma^2}{2} \right) t + \sigma W_t \right)$$

Parameters

- **drift** (*float*) – the parameter μ

- **volatility** (*float*) – the parameter σ
- **t** (*float*) – the right hand endpoint of the time interval $[0, t]$ for the process
- **rng** (*numpy.random.Generator*) – a custom random number generator

property drift

Geometric Brownian motion drift parameter.

sample(*n*, *initial=1*)

Generate a realization.

Parameters

- **n** (*int*) – the number of increments to generate.
- **initial** (*float*) – the initial value of the process S_0.

sample_at(*times*, *initial=1*)

Generate a realization using specified times.

Parameters

- **times** – a vector of increasing time values at which to generate the realization
- **initial** (*float*) – the initial value of the process S_0.

property t

End time of the process.

times(*n*)

Generate times associated with n increments on [0, t].

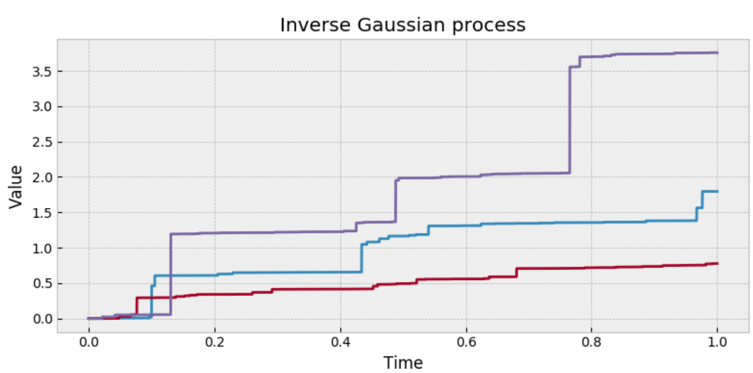
Parameters n (*int*) – the number of increments

property volatility

Geometric Brownian motion volatility parameter.

class stochastic.processes.continuous.**InverseGaussianProcess**(*mean=None*, *scale=1*, *t=1*, *rng=None*)

Inverse Gaussian process.



An inverse Gaussian process has independent increments which follow an inverse Gaussian distribution with parameters defined by a monotonically increasing function, $\Gamma(t)$. E.g. for increment $[s, t]$:

$\mathcal{IG}(\Gamma(t) - \Gamma(s), \eta(\Gamma(t) - \Gamma(s))^2)$

Uses a method for generating inverse Gaussian variates from:

- Michael, John R., William R. Schucany, and Roy W. Haas. "Generating random variates using transformations with multiple roots." The American Statistician 30, no. 2 (1976): 88-90.

Parameters

- **mean** (*callable*) – a callable with one argument $\Gamma(t)$ such that $\Gamma(t') > \Gamma(t) \forall t' > t$. Default is the identity function.
- **scale** (*float*) – scale factor of the shape parameter of the inverse gaussian, or η from the above equation.
- **t** (*float*) – the right hand endpoint of the time interval $[0, t]$ for the process
- **rng** (*numpy.random.Generator*) – a custom random number generator

`property mean`
Mean function.

`sample(n)`
Generate a realization.

Parameters **n** (*int*) – the number of increments to generate

`sample_at(times)`
Generate a realization using specified times.

Parameters **times** – a vector of increasing time values at which to generate the realization

`property scale`
Scale parameter.

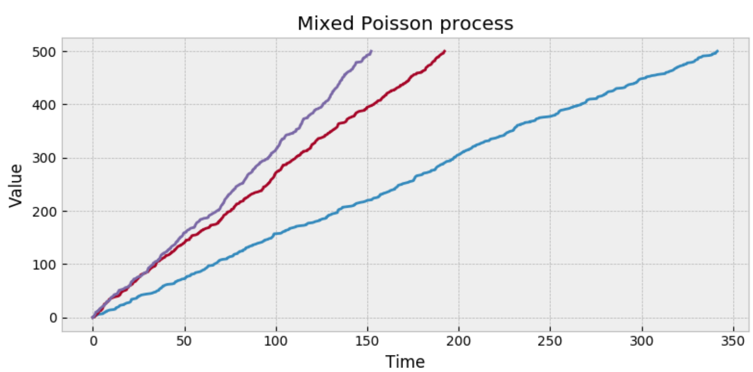
`property t`
End time of the process.

`times(n)`
Generate times associated with n increments on [0, t].

Parameters **n** (*int*) – the number of increments

`class stochastic.processes.continuous.MixedPoissonProcess(rate_func, rate_args=None, rate_kwargs=None, rng=None)`

Mixed poisson process.



A mixed poisson process is a Poisson process for which the rate is a scalar random variate. The sample method will generate a random variate for the rate before generating a Poisson process realization with the rate. A Poisson process with rate λ is a count of occurrences of i.i.d. exponential random variables with mean $1/\lambda$. Use the `rate` attribute to get the most recently generated random rate.

Parameters

- **rate_func** (*callable*) – a callable to generate variates of the random rate
- **rate_args** (*tuple*) – positional args for `rate_func`

- **rate_kwargs** (*dict*) – keyword args for `rate_func`
- **rng** (*numpy.random.Generator*) – a custom random number generator

property rate
The most recently generated rate.

Attempting to get the rate prior to generating a sample will raise an `AttributeError`.

property rate_args
Positional arguments for the rate function.

property rate_func
Current rate's distribution.

property rate_kwargs
Keyword arguments for the rate function.

sample(*n=None*, *length=None*)
Generate a realization.

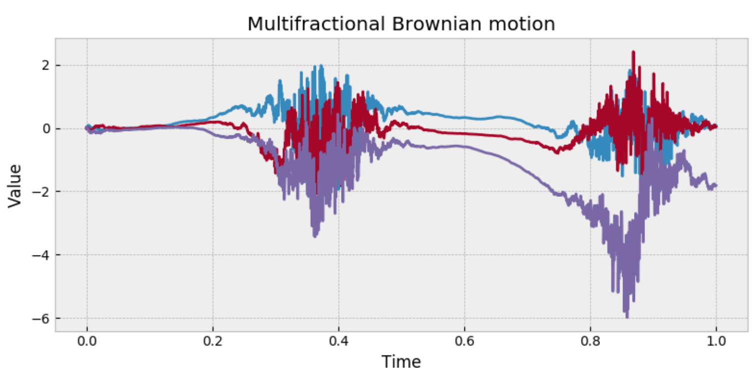
Exactly one of *n* and *length* must be provided. Generates a random variate for the rate, then generates a Poisson process realization using this rate.

Parameters

- **n** (*int*) – the number of arrivals to simulate
- **length** (*int*) – the length of time to simulate; will generate arrivals until length is met or exceeded.

class stochastic.processes.continuous.**MultifractionalBrownianMotion**(*hurst=None*, *t=1*, *rng=None*)

Multifractional Brownian motion process.



A multifractional Brownian motion generalizes a fractional Brownian motion with a Hurst parameter which is a function of time, $h(t)$. If the Hurst is constant, the process is a fractional Brownian motion. If Hurst is constant equal to 0.5, the process is a Brownian motion.

Approximate method originally proposed for fBm in

- Rambaldi, Sandro, and Ombretta Pinazza. "An accurate fractional Brownian motion generator." Physica A: Statistical Mechanics and its Applications 208, no. 1 (1994): 21-30.

Adapted to approximate mBm in

- Muniandy, S. V., and S. C. Lim. "Modeling of locally self-similar processes using multifractional Brownian motion of Riemann-Liouville type." Physical Review E 63, no. 4 (2001): 046104.

Parameters

- **hurst** (*float*) – a callable with one argument $h(t)$ such that $h(t') \in (0,1) \forall t' \in [0,t]$. Default is $h(t) = 0.5$.
- **t** (*float*) – the right hand endpoint of the time interval $[0,t]$ for the process
- **rng** (*numpy.random.Generator*) – a custom random number generator

property hurst
Hurst function.

sample(*n*)
Generate a realization.

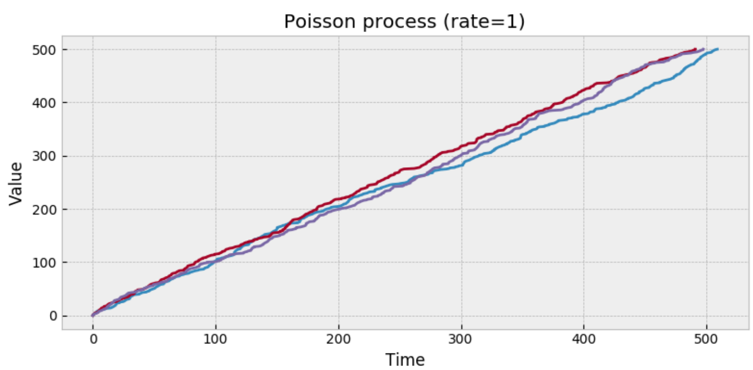
Parameters n (*int*) – the number of increments to generate

property t
End time of the process.

times(*n*)
Generate times associated with n increments on [0, t].

Parameters n (*int*) – the number of increments

class stochastic.processes.continuous.**PoissonProcess**(*rate=1*, *rng=None*)
Poisson process.



A Poisson process with rate λ is a count of occurrences of i.i.d. exponential random variables with mean $1/\lambda$. This class generates samples of times for which cumulative exponential random variables occur.

Parameters

- **rate** (*float*) – the parameter λ which defines the rate of occurrences of the process
- **rng** (*numpy.random.Generator*) – a custom random number generator

property rate
Rate parameter.

sample(*n=None*, *length=None*)
Generate a realization.

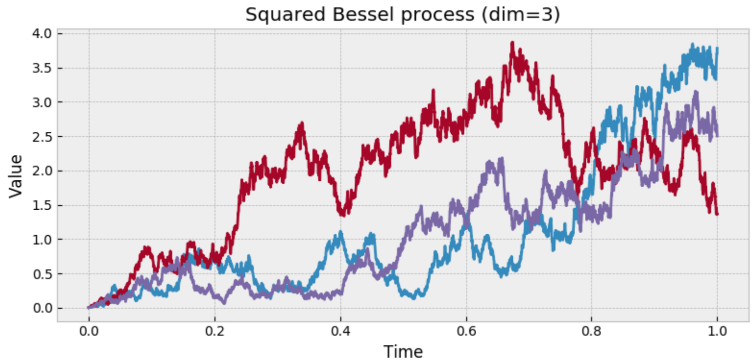
Exactly one of *n* and *length* must be provided.

Parameters

- **n** (*int*) – the number of arrivals to simulate
- **length** (*int*) – the length of time to simulate; will generate arrivals until length is met or exceeded.

class stochastic.processes.continuous.**SquaredBesselProcess**(*dim=1*, *t=1*, *rng=None*)

Squared Bessel process.



The square of a Bessel process: $\|\mathbf{W}_t\|^2$.

The Bessel process is the Euclidean norm of an n-dimensional Wiener process, e.g. $\|\mathbf{W}_t\|$

Parameters

- **dim** (*int*) – the number of underlying independent Brownian motions to use
- **t** (*float*) – the right hand endpoint of the time interval $[0, t]$ for the process

property dim

Dimensions, or independent Brownian motions.

sample(*n*)

Generate a realization.

Parameters **n** (*int*) – the number of increments to generate

sample_at(*times*)

Generate a realization using specified times.

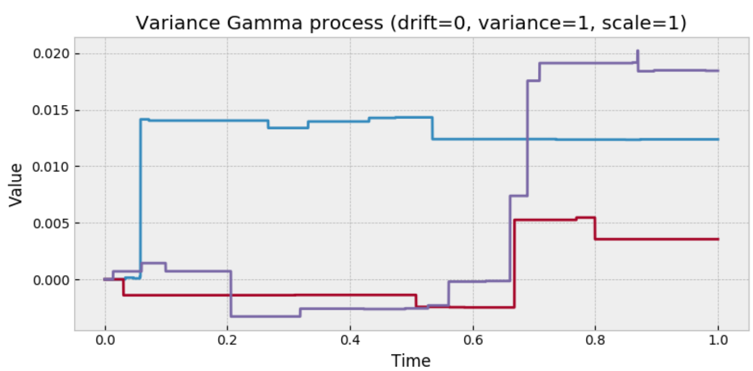
Parameters **times** – a vector of increasing time values at which to generate the realization

property t

End time of the process.

class stochastic.processes.continuous.**VarianceGammaProcess**(*drift=0*, *variance=1*, *scale=1*, *t=1*, *rng=None*)

Variance Gamma process.



A variance gamma process has independent increments which follow the variance-gamma distribution. It can be represented as a Brownian motion with drift subordinated by a Gamma process:

$$\theta\Gamma(t; 1, \nu) + \sigma W(\Gamma(t; 1, \nu))$$

Parameters

- **drift** (*float*) – the drift parameter of the Brownian motion, or θ above
- **variance** (*float*) – the variance parameter of the Gamma subordinator, or ν above
- **scale** (*float*) – the scale parameter of the Brownian motion, or σ above
- **t** (*float*) – the right hand endpoint of the time interval $[0, t]$ for the process
- **rng** (*numpy.random.Generator*) – a custom random number generator

property drift
Drift parameter.

sample(*n*)
Generate a realization.

Parameters **n** (*int*) – the number of increments to generate

sample_at(*times*)
Generate a realization using specified times.

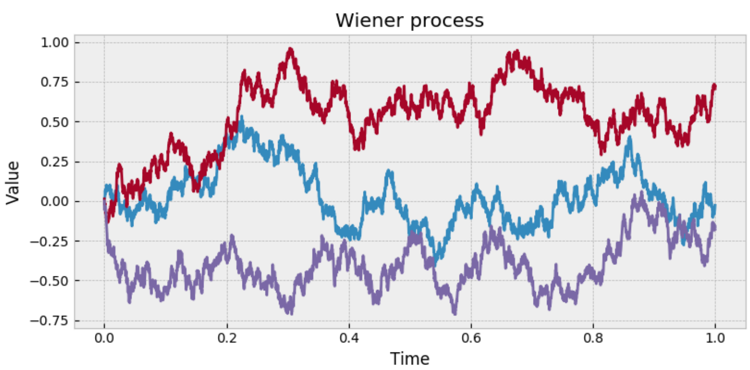
Parameters **times** – a vector of increasing time values at which to generate the realization

property scale
Scale parameter.

property t
End time of the process.

property variance
Variance parameter.

class stochastic.processes.continuous.**WienerProcess**(*t=1*, *rng=None*)
Wiener process, or standard Brownian motion.



Parameters

- **t** (*float*) – the right hand endpoint of the time interval $[0, t]$ for the process
- **rng** (*numpy.random.Generator*) – a custom random number generator

sample(*n*)
Generate a realization.

Parameters **n** (*int*) – the number of increments to generate

sample_at(*times*)
Generate a realization using specified times.

Parameters **times** – a vector of increasing time values at which to generate the realization

property t
End time of the process.

times(*n*)
Generate times associated with n increments on [0, t].

Parameters **n** (*int*) – the number of increments

5.4 Diffusion Models

The `stochastic.processes.diffusion` module provides classes for generating discretely sampled continous-time diffusion processes using the Euler–Maruyama method.

- *stochastic.processes.diffusion.DiffusionProcess*
- *stochastic.processes.diffusion.ConstantElasticityVarianceProcess*
- *stochastic.processes.diffusion.CoxIngersollRossProcess*
- *stochastic.processes.diffusion.OrnsteinUhlenbeckProcess*
- *stochastic.processes.diffusion.VasicekProcess*

class stochastic.processes.diffusion.**DiffusionProcess**(*speed=1*, *mean=0*, *vol=1*, *volexp=0*, *t=1*, *rng=None*)

Generalized diffusion process.

A base process for more specific diffusion processes.

The process X_t that satisfies the following stochastic differential equation with Wiener process W_t:

$$dX_t = \theta_t(\mu_t - X_t)dt + \sigma_t X_t^{\gamma_t} dW_t$$

Realizations are generated using the Euler-Maruyama method.

Note: Since the family of diffusion processes have parameters which generalize to functions of `t`, parameter attributes will be returned as callables, even if they are initialized as constants. e.g. a `speed` parameter of 1 accessed from an instance attribute will return a function which accepts a single argument and always returns 1.

Parameters

- **speed** (*func*) – the speed of reversion, or θ_t above
- **mean** (*func*) – the mean of the process, or μ_t above
- **vol** (*func*) – volatility coefficient of the process, or σ_t above
- **volexp** (*func*) – volatility exponent of the process, or γ_t above
- **t** (*float*) – the right hand endpoint of the time interval $[0, t]$ for the process
- **rng** (*numpy.random.Generator*) – a custom random number generator

`sample`(*n*, *initial=1.0*)
Generate a realization.

Parameters

- **n** (*int*) – the number of increments to generate
- **initial** (*float*) – the initial value of the process

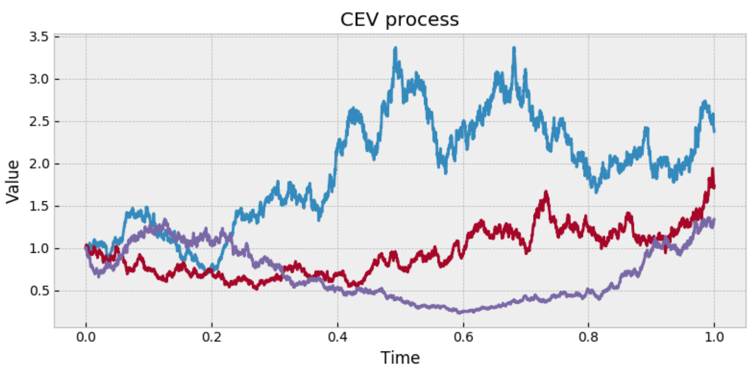
property t
End time of the process.

`times`(*n*)
Generate times associated with n increments on [0, t].

Parameters **n** (*int*) – the number of increments

class stochastic.processes.diffusion.**ConstantElasticityVarianceProcess**(*drift=1*, *vol=1*, *volexp=1*, *t=1*, *rng=None*)

Constant elasticity of variance process.



The process X_t that satisfies the following stochastic differential equation with Wiener process W_t:

$$dX_t = \mu X_t dt + \sigma X_t^\gamma dW_t$$

Realizations are generated using the Euler-Maruyama method.

Note: Since the family of diffusion processes have parameters which generalize to functions of `t`, parameter attributes will be returned as callables, even if they are initialized as constants. e.g. a `speed` parameter of 1 accessed from an instance attribute will return a function which accepts a single argument and always returns 1.

Parameters

- **drift** (*float*) – the drift coefficient, or μ above
- **vol** (*float*) – the volatility coefficient, or σ above
- **volexp** (*float*) – the volatility-price exponent, or γ above
- **t** (*float*) – the right hand endpoint of the time interval $[0, t]$ for the process
- **rng** (*numpy.random.Generator*) – a custom random number generator

`sample`(*n*, *initial=1.0*)
Generate a realization.

Parameters

- **n** (*int*) – the number of increments to generate
- **initial** (*float*) – the initial value of the process

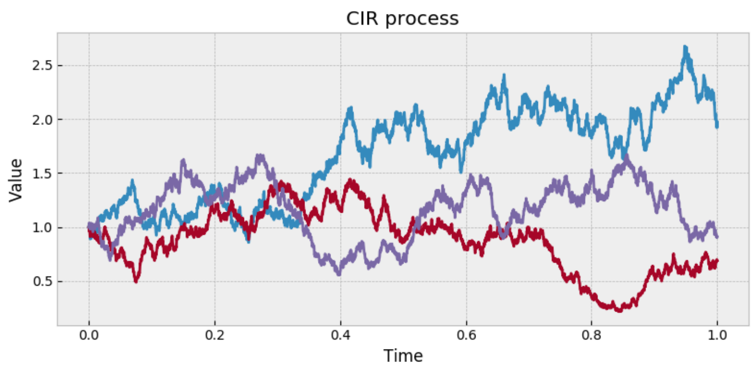
property t
End time of the process.

times(*n*)
Generate times associated with n increments on [0, t].

Parameters **n** (*int*) – the number of increments

class stochastic.processes.diffusion.**CoxIngersollRossProcess**(*speed=1*, *mean=0*, *vol=1*, *t=1*, *rng=None*)

Cox-Ingersoll-Ross process.



A model for instantaneous interest rate.

The process X_t that satisfies the following stochastic differential equation with Wiener process W_t:

$$dX_t = \theta(\mu - X_t)dt + \sigma\sqrt{X_t}dW_t$$

Realizations are generated using the Euler-Maruyama method.

Note: Since the family of diffusion processes have parameters which generalize to functions of `t`, parameter attributes will be returned as callables, even if they are initialized as constants. e.g. a `speed` parameter of 1 accessed from an instance attribute will return a function which accepts a single argument and always returns 1.

Parameters

- **speed** (*float*) – the speed of reversion, or θ above
- **mean** (*float*) – the mean of the process, or μ above
- **vol** (*float*) – volatility coefficient of the process, or σ above
- **t** (*float*) – the right hand endpoint of the time interval $[0, t]$ for the process
- **rng** (*numpy.random.Generator*) – a custom random number generator

sample(*n*, *initial=1.0*)
Generate a realization.

Parameters

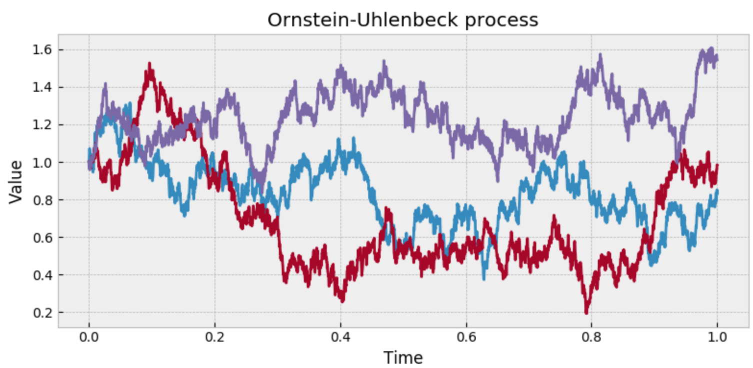
- **n** (*int*) – the number of increments to generate
- **initial** (*float*) – the initial value of the process

property t
End time of the process.

times(*n*)
Generate times associated with n increments on [0, t].

Parameters **n** (*int*) – the number of increments

class stochastic.processes.diffusion.**OrnsteinUhlenbeckProcess**(*speed=1*, *vol=1*, *t=1*, *rng=None*)
Ornstein-Uhlenbeck process.



The process X_t that satisfies the following stochastic differential equation with Wiener process W_t:

$$dX_t = -\theta X_t dt + \sigma dW_t$$

Realizations are generated using the Euler-Maruyama method.

Note: Since the family of diffusion processes have parameters which generalize to functions of `t`, parameter attributes will be returned as callables, even if they are initialized as constants. e.g. a `speed` parameter of 1 accessed from an instance attribute will return a function which accepts a single argument and always returns 1.

Parameters

- **speed** (*float*) – the speed of reversion, or θ above
- **vol** (*float*) – volatility coefficient of the process, or σ above
- **t** (*float*) – the right hand endpoint of the time interval $[0, t]$ for the process
- **rng** (*numpy.random.Generator*) – a custom random number generator

sample(*n*, *initial=1.0*)
Generate a realization.

Parameters

- **n** (*int*) – the number of increments to generate
- **initial** (*float*) – the initial value of the process

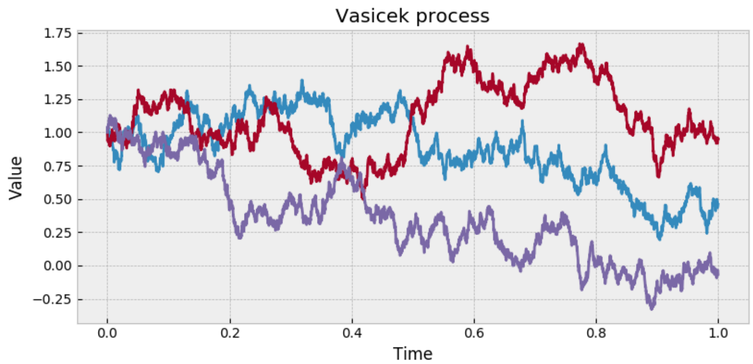
property t
End time of the process.

times(*n*)
Generate times associated with n increments on [0, t].

Parameters **n** (*int*) – the number of increments

class stochastic.processes.diffusion.**VasicekProcess**(*speed=1*, *mean=1*, *vol=1*, *t=1*, *rng=None*)
Vasicek process.

A model for instantaneous interest rate.



The Vasicek process X_t that satisfies the following stochastic differential equation with Wiener process W_t:

$$dX_t = \theta(\mu - X_t)dt + \sigma dW_t$$

Realizations are generated using the Euler-Maruyama method.

Note: Since the family of diffusion processes have parameters which generalize to functions of `t`, parameter attributes will be returned as callables, even if they are initialized as constants. e.g. a `speed` parameter of 1 accessed from an instance attribute will return a function which accepts a single argument and always returns 1.

Parameters

- **speed** (*float*) – the speed of reversion, or θ above
- **mean** (*float*) – the mean of the process, or μ above
- **vol** (*float*) – volatility coefficient of the process, or σ above
- **t** (*float*) – the right hand endpoint of the time interval $[0, t]$ for the process
- **rng** (*numpy.random.Generator*) – a custom random number generator

sample(*n*, *initial=1.0*)
Generate a realization.

Parameters

- **n** (*int*) – the number of increments to generate
- **initial** (*float*) – the initial value of the process

property t
End time of the process.

times(*n*)
Generate times associated with n increments on [0, t].

Parameters **n** (*int*) – the number of increments

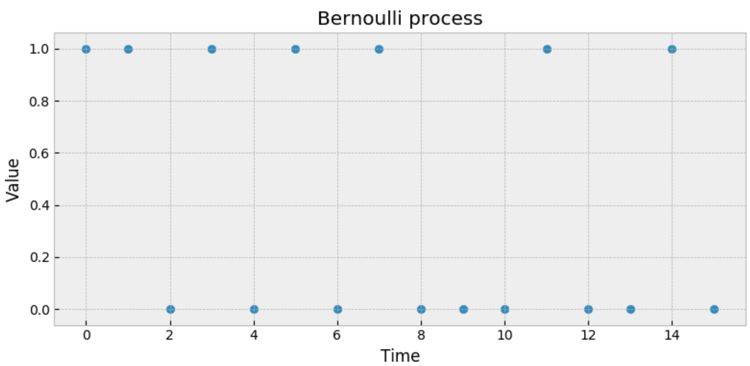
5.5 Discrete-time Processes

The `stochastic.processes.discrete` module provides classes for generating discrete-time stochastic processes.

- *stochastic.processes.discrete.BernoulliProcess*
- *stochastic.processes.discrete.ChineseRestaurantProcess*
- *stochastic.processes.discrete.DirichletProcess*
- *stochastic.processes.discrete.MarkovChain*
- *stochastic.processes.discrete.MoranProcess*
- *stochastic.processes.discrete.RandomWalk*

class stochastic.processes.discrete.**BernoulliProcess**(*p=0.5*, *rng=None*)

Bernoulli process.



A Bernoulli process consists of a sequence of Bernoulli random variables. A Bernoulli random variable is

- 1 with probability p
- 0 with probaiility $1 - p$

Parameters

- **p** – in $[0, 1]$, the probability of success of each Bernoulli random variable
- **rng** (*numpy.random.Generator*) – a custom random number generator

property p

Probability of success.

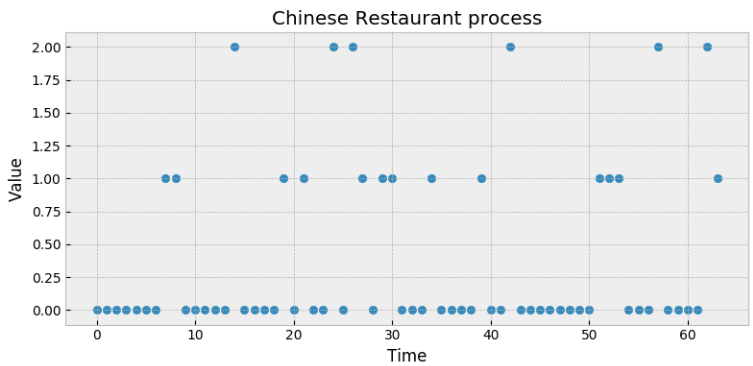
sample(*n*)

Generate a Bernoulli process realization.

Parameters **n** (*int*) – the number of steps to simulate.

class stochastic.processes.discrete.**ChineseRestaurantProcess**(*discount=0*, *strength=1*, *rng=None*)

Chinese restaurant process.



A Chinese restaurant process consists of a sequence of arrivals of customers to a Chinese restaurant. Customers may be seated either at an occupied table or a new table, there being infinitely many customers and tables.

The first customer sits at the first table. The n-th customer sits at a new table with probability $1/n$, and at each already occupied table with probability t_k/n, where t_k is the number of customers already seated at table k. This is the canonical process with $discount = 0$ and $strength = 1$.

The generalized process gives the n-th customer a probability of $(strength + T * discount)/(n - 1 + strength)$ to sit at a new table and a probability of $(t_k - discount)/(n - 1 + strength)$ of sitting at table k. T is the number of occupied tables.

Samples provide a sequence of tables selected by a sequence of customers.

Parameters

- **discount** (*float*) – the discount value of existing tables. Must be strictly less than 1.
- **strength** (*float*) – the strength of a new table. If discount is negative, strength must be a multiple of discount. If discount is nonnegative, strength must be strictly greater than the negative discount.
- **rng** (*numpy.random.Generator*) – a custom random number generator

property discount
Discount parameter.

partition_to_sequence(*partition*)
Create a sequence from a partition.

Parameters **partition** – a Chinese restaurant partition.

sample(*n*)
Generate a Chinese restaurant process with n customers.

Parameters **n** – the number of customers to simulate.

sample_partition(*n*)
Generate a Chinese restaurant process partition.

Parameters **n** – the number of customers to simulate.

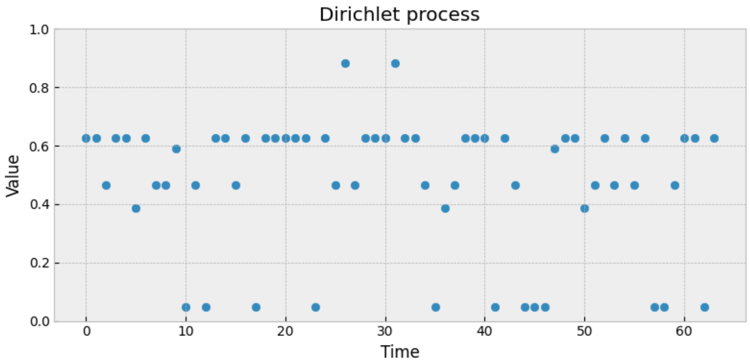
sequence_to_partition(*sequence*)
Create a partition from a sequence.

Parameters **sequence** – a Chinese restaurant sample.

property strength
Strength parameter.

class stochastic.processes.discrete.**DirichletProcess**(*base=None*, *alpha=1*, *rng=None*)

Dirichlet process.



A Dirichlet process is a stochastic process in which the resulting samples can be interpreted as discrete probability distributions.

For each step $k \geq 1$, draw from the base distribution with probability

$$\frac{\alpha}{\alpha + k - 1}$$

Otherwise draw randomly from the previous steps.

Parameters

- **base** (*callable*) – a zero argument callable used as the base distribution sampler. The default base distribution is Uniform(0, 1).
- **alpha** (*float*) – a non-negative value used to determine probability of drawing a new value from the base distribution
- **rng** (*numpy.random.Generator*) – a custom random number generator

property alpha

Parameter for determining the probability of sampling new values.

property base

The base distribution callable for sampling new step values.

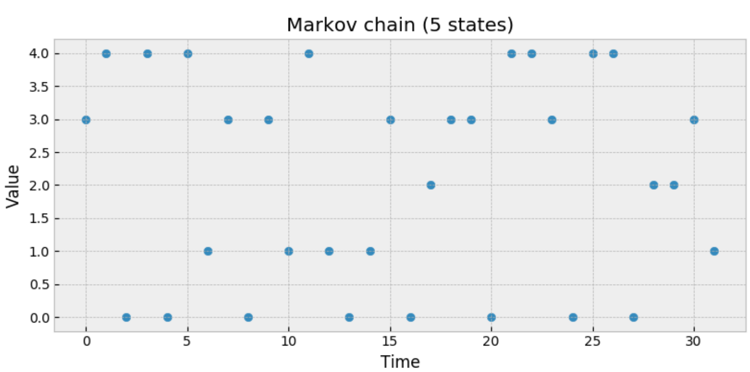
sample(*n*)

Generate a realization of the Dirichlet process.

Parameters n (*int*) – the number of steps of the Dirichlet process to generate.

class stochastic.processes.discrete.**MarkovChain**(*transition=None*, *initial=None*, *rng=None*)

Finite state Markov chain.



A Markov Chain which changes between states according to the transition matrix.

Parameters

- **transition** (*2darray*) – a square matrix representing the transition probabilities between states.
- **initial** (*1darray*) – a vector representing the initial state probabilities. If not provided, each state has equal initial probability.
- **rng** (*numpy.random.Generator*) – a custom random number generator

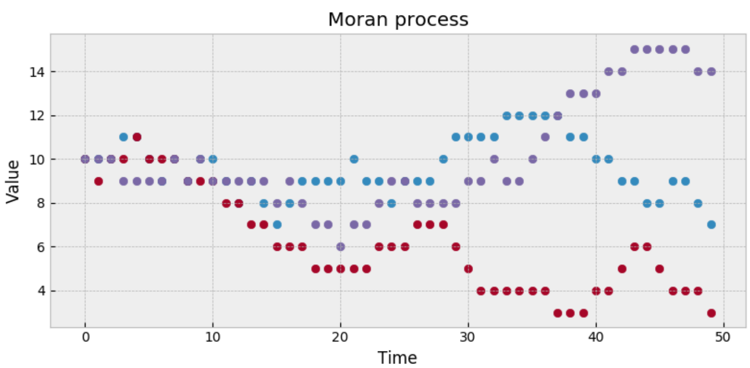
property initial
Vector of initial state probabilities.

sample(*n*)
Generate a realization of the Markov chain.

Parameters n (*int*) – the number of steps of the Markov chain to generate.

property transition
Transition probability matrix.

class stochastic.processes.discrete.**MoranProcess**(*maximum*, *rng=None*)
Moran process.



A neutral drift Moran process, typically used to model populations. At each step this process will increase by one, decrease by one, or remain at the same value between values of zero and the number of states, n. The process ends when its value reaches zero or the maximum valued state.

Parameters

- **maximum** (*int*) – the maximum possible value for the process.
- **rng** (*numpy.random.Generator*) – a custom random number generator

property maximum
Maximum value.

sample(*n*, *start*)
Generate a realization of the Moran process.

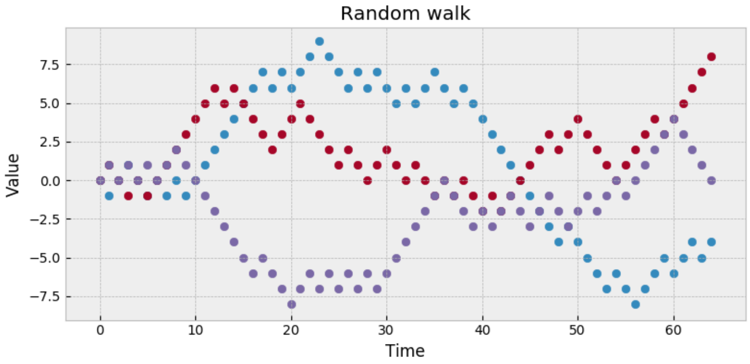
Generate a Moran process until absorption occurs (state 0 or `maximum`) or length of process reaches length n.

Parameters

- **n** (*int*) – the maximum number of steps to generate assuming absorption does not occur.
- **start** (*int*) – the initial state of the process.

class stochastic.processes.discrete.**RandomWalk**(*steps=None*, *weights=None*, *rng=None*)

Random walk.



A random walk is a sequence of random steps taken from a set of step sizes with a probability distribution. By default this object defines the steps to be [-1, 1] with probability 1/2 for each possibility.

Parameters

- **steps** – a vector of possible deltas to apply at each step.
- **weights** – a corresponding vector of weights associated with each step value. If not provided each step has equal weight/probability.

property p

Step probabilities, normalized from *weights*.

sample(*n*)

Generate a sample random walk.

Parameters n (*int*) – the number of steps to generate

sample_increments(*n*)

Generate a sample of random walk increments.

Parameters n (*int*) – the number of increments to generate.

property steps

Possible steps.

property weights

Step weights provided.

5.6 Noise Processes

The stochastic.processes.noise module provides classes for generating noise processes.

Gaussian increments

- *stochastic.processes.noise.GaussianNoise*
- *stochastic.processes.noise.FractionalGaussianNoise*

Colored noise

- *stochastic.processes.noise.BlueNoise*
- *stochastic.processes.noise.BrownianNoise*
- *stochastic.processes.noise.ColoredNoise*

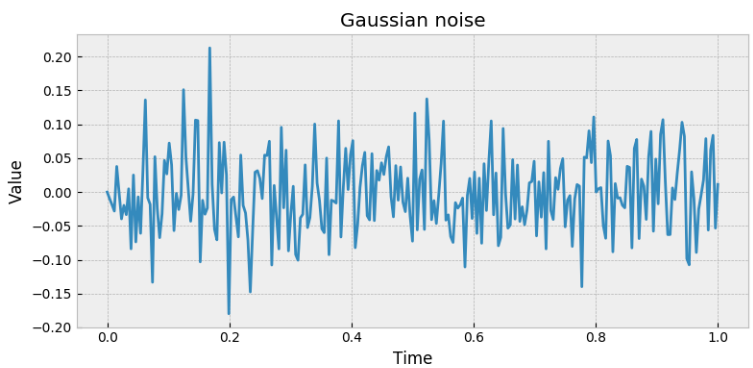
- *stochastic.processes.noise.RedNoise*
- *stochastic.processes.noise.PinkNoise*
- *stochastic.processes.noise.VioletNoise*
- *stochastic.processes.noise.WhiteNoise*

5.6.1 Gaussian increments

Noise processes which are increments of their continuous counterparts.

class stochastic.processes.noise.**GaussianNoise**(*t=1*, *rng=None*)

Gaussian noise process.



Generate a sequence of Gaussian random variables.

Parameters

- **t** (*float*) – the right hand endpoint of the time interval $[0, t]$ for the process
- **rng** (*numpy.random.Generator*) – a custom random number generator

sample(*n*)

Generate a realization of Gaussian noise.

Generate a Gaussian noise realization with n increments.

Parameters **n** (*int*) – the number of increments to generate.

sample_at(*times*)

Generate Gaussian noise increments at specified times from zero.

Parameters **times** – a vector of increasing time values for which to generate noise increments.

property t

End time of the process.

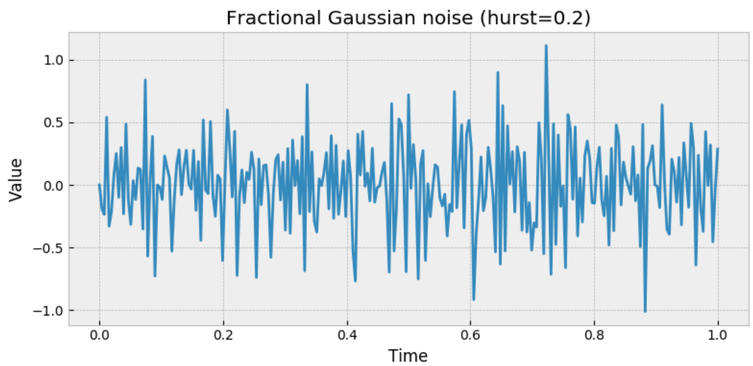
times(*n*)

Generate times associated with n increments on [0, t].

Parameters **n** (*int*) – the number of increments

class stochastic.processes.noise.**FractionalGaussianNoise**(*hurst=0.5*, *t=1*, *rng=None*)

Fractional Gaussian noise process.



Generate sequences of fractional Gaussian noise.

Hosking's method:

- Hosking, Jonathan RM. "Modeling persistence in hydrological time series using fractional differencing." Water resources research 20, no. 12 (1984): 1898-1908.

Davies Harte method:

- Davies, Robert B., and D. S. Harte. "Tests for Hurst effect." Biometrika 74, no. 1 (1987): 95-101.

Parameters

- **hurst** (*float*) – The Hurst parameter value in $(0, 1)$.
- **t** (*float*) – the right hand endpoint of the time interval $[0, t]$ for the process
- **rng** (*numpy.random.Generator*) – a custom random number generator

property hurst

Hurst parameter.

sample(*n*, *algorithm='daviesharte'*)

Generate a realization of fractional Gaussian noise.

Parameters

- **n** (*int*) – number of increments to generate
- **algorithm** (*str*) – either 'daviesharte' or 'hosking' algorithms

property t

End time of the process.

times(*n*)

Generate times associated with n increments on [0, t].

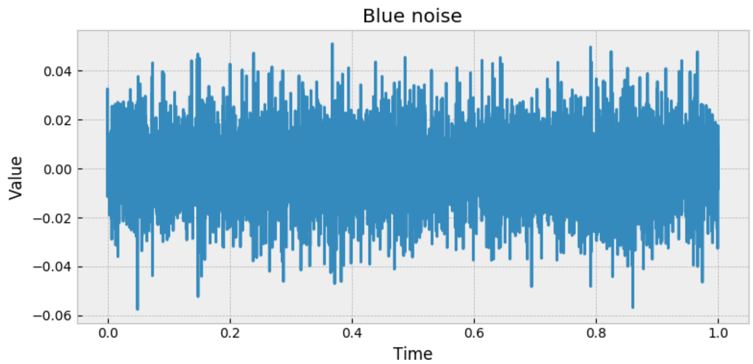
Parameters **n** (*int*) – the number of increments

5.6.2 Colored noise

Signals with spectral densities proportional to the power law.

class `stochastic.processes.noise.`**`BlueNoise`**(*t=1*, *rng=None*)

Blue noise.



Colored noise, or power law noise with spectral density exponent $\beta = -1$.

Parameters

- **t** (*float*) – the right hand endpoint of the time interval $[0, t]$ for the process
- **rng** (*numpy.random.Generator*) – a custom random number generator

`sample`(*n*)

Generate a realization of colored noise.

Generate a colored noise realization with n increments.

Parameters **n** (*int*) – the number of increments to generate.

property t

End time of the process.

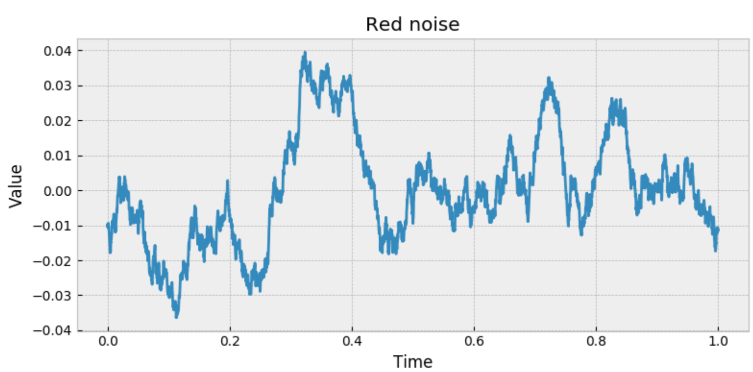
`times`(*n*)

Generate times associated with n increments on [0, t].

Parameters **n** (*int*) – the number of increments

class `stochastic.processes.noise.`**`BrownianNoise`**(*t=1*, *rng=None*)

Brownian (red) noise.



Colored noise, or power law noise with spectral density exponent $\beta = 2$.

Parameters

- **t** (*float*) – the right hand endpoint of the time interval $[0, t]$ for the process
- **rng** (*numpy.random.Generator*) – a custom random number generator

`sample(n)`
Generate a realization of colored noise.

Generate a colored noise realization with n increments.

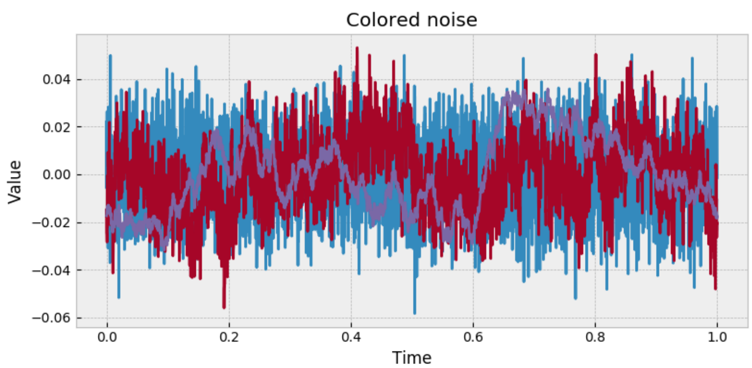
Parameters **n** (*int*) – the number of increments to generate.

`property t`
End time of the process.

`times(n)`
Generate times associated with n increments on [0, t].

Parameters **n** (*int*) – the number of increments

`class stochastic.processes.noise.ColoredNoise(beta=0, t=1, rng=None)`
Colored noise processes.



Also referred to as power law noise, colored noise refers to noise processes with power law spectral density. That is, their spectral density per unit bandwidth is proportional to $(1/f)^{\beta}$, where f is frequency with exponent β.

Uses the algorithm from:

- Timmer, J., and M. Koenig. "On generating power law noise." Astronomy and Astrophysics 300 (1995): 707.

Generates a normalized power-law spectral noise.

Parameters

- **beta** (*float*) – the power law exponent for the spectral density, with 0 being white noise, 1 being pink noise, 2 being red noise (Brownian noise), -1 being blue noise, -2 being violet noise. Default is 0 (white noise).
- **t** (*float*) – the right hand endpoint of the time interval $[0, t]$ for the process
- **rng** (*numpy.random.Generator*) – a custom random number generator

`property beta`
Power law exponent.

`sample(n)`
Generate a realization of colored noise.

Generate a colored noise realization with n increments.

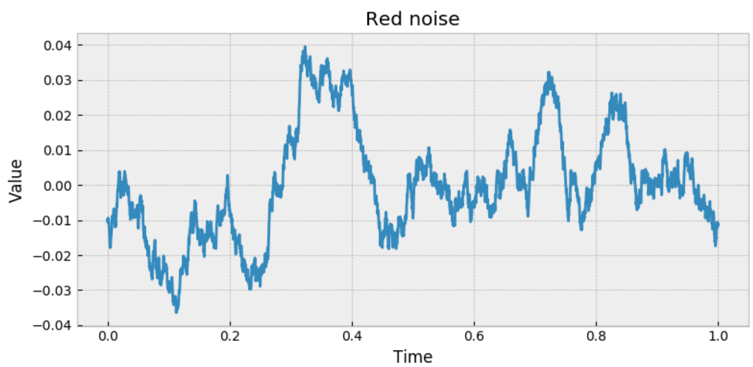
Parameters **n** (*int*) – the number of increments to generate.

property t
End time of the process.

times(*n*)
Generate times associated with n increments on [0, t].

Parameters **n** (*int*) – the number of increments

class stochastic.processes.noise.**RedNoise**(*t=1*, *rng=None*)
Red (Brownian) noise.



Colored noise, or power law noise with spectral density exponent $\beta = 2$.

Parameters

- **t** (*float*) – the right hand endpoint of the time interval $[0, t]$ for the process
- **rng** (*numpy.random.Generator*) – a custom random number generator

sample(*n*)
Generate a realization of colored noise.

Generate a colored noise realization with n increments.

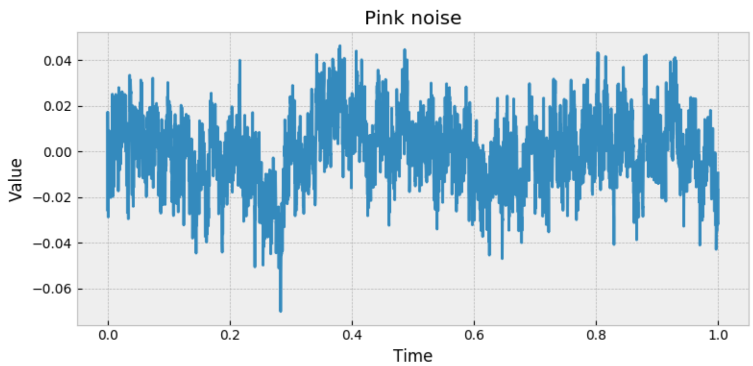
Parameters **n** (*int*) – the number of increments to generate.

property t
End time of the process.

times(*n*)
Generate times associated with n increments on [0, t].

Parameters **n** (*int*) – the number of increments

class stochastic.processes.noise.**PinkNoise**(*t=1*, *rng=None*)
Pink (flicker) noise.



Colored noise, or power law noise with spectral density exponent $\beta = 1$.

Parameters

- **t** (*float*) – the right hand endpoint of the time interval $[0, t]$ for the process
- **rng** (*numpy.random.Generator*) – a custom random number generator

sample(*n*)
Generate a realization of colored noise.

Generate a colored noise realization with n increments.

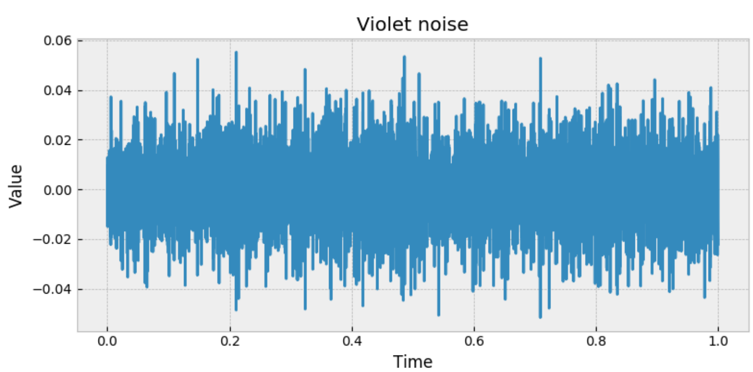
Parameters **n** (*int*) – the number of increments to generate.

property t
End time of the process.

times(*n*)
Generate times associated with n increments on [0, t].

Parameters **n** (*int*) – the number of increments

class stochastic.processes.noise.**VioletNoise**(*t=1*, *rng=None*)
Violet noise.



Colored noise, or power law noise with spectral density exponent $\beta = -2$.

Parameters

- **t** (*float*) – the right hand endpoint of the time interval $[0, t]$ for the process
- **rng** (*numpy.random.Generator*) – a custom random number generator

sample(*n*)
Generate a realization of colored noise.

Generate a colored noise realization with n increments.

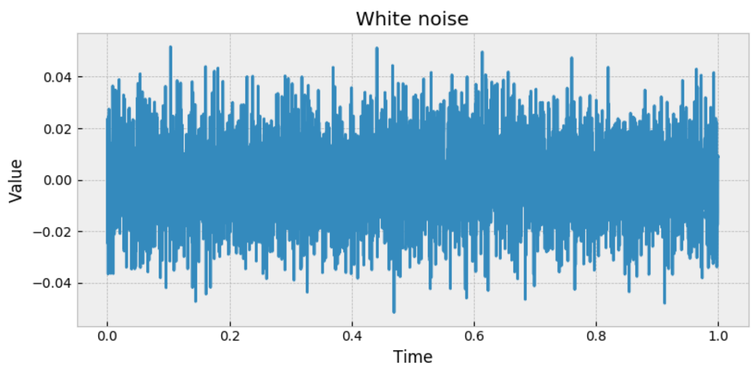
Parameters **n** (*int*) – the number of increments to generate.

property t
End time of the process.

times(*n*)
Generate times associated with n increments on [0, t].

Parameters **n** (*int*) – the number of increments

class stochastic.processes.noise.**WhiteNoise**(*t=1*, *rng=None*)
White noise.



Colored noise, or power law noise with spectral density exponent $\beta = 0$.

Parameters

- **t** (*float*) – the right hand endpoint of the time interval $[0, t]$ for the process
- **rng** (*numpy.random.Generator*) – a custom random number generator

sample(*n*)
Generate a realization of colored noise.

Generate a colored noise realization with n increments.

Parameters **n** (*int*) – the number of increments to generate.

property t
End time of the process.

times(*n*)
Generate times associated with n increments on [0, t].

Parameters **n** (*int*) – the number of increments

5.7 Bibliographical Sources

- Asmussen, Søren. Stochastic simulation with a view towards stochastic processes. University of Aarhus. Centre for Mathematical Physics and Stochastics (MaPhySto)[MPS], 1998.
- Biane, Philippe. “Relations entre pont et excursion du mouvement Brownien réel.” Ann. Inst. Henri Poincaré 22, no. 1 (1986): 1-7.
- Davies, Robert B., and D. S. Harte. “Tests for Hurst effect.” Biometrika 74, no. 1 (1987): 95-101.

- Devroye, Luc. "On exact simulation algorithms for some distributions related to Brownian motion and Brownian meanders." In Recent developments in applied probability and statistics, pp. 1-35. Physica-Verlag HD, 2010.
- Hosking, Jonathan RM. "Modeling persistence in hydrological time series using fractional differencing." Water resources research 20, no. 12 (1984): 1898-1908.
- Imhof, J-P. "Density factorizations for Brownian motion, meander and the three-dimensional Bessel process, and applications." Journal of Applied Probability 21, no. 3 (1984): 500-510.
- Michael, John R., William R. Schucany, and Roy W. Haas. "Generating random variates using transformations with multiple roots." The American Statistician 30, no. 2 (1976): 88-90.
- Muniandy, S. V., and S. C. Lim. "Modeling of locally self-similar processes using multifractional Brownian motion of Riemann-Liouville type." Physical Review E 63, no. 4 (2001): 046104.
- Rambaldi, Sandro, and Ombretta Pinazza. "An accurate fractional Brownian motion generator." Physica A: Statistical Mechanics and its Applications 208, no. 1 (1994): 21-30.
- Timmer, J., and M. Koenig. "On generating power law noise." Astronomy and Astrophysics 300 (1995): 707.
- Vervaat, Wim. "A relation between Brownian bridge and Brownian excursion." The Annals of Probability (1979): 143-149.
- Williams, David. "Decomposing the Brownian path." Bulletin of the American Mathematical Society 76, no. 4 (1970): 871-873.
- Wood, Andrew TA, and Grace Chan. "Simulation of stationary Gaussian processes in [0, 1] d." Journal of computational and graphical statistics 3, no. 4 (1994): 409-432.

5.8 Release Notes

5.8.1 Contributing

Stochastic is an open source python package.

If you have additional processes, generalizations, or algorithms that you think would be suitable for this package, please let me know on this project's GitHub page.

5.8.2 License

(continues on next page)

(continued from previous page)

5.8.3 Release Changelog

0.7.0 (2022-07-11)

- Don't install meta assets to `site-packages` folder
- Pass GBM `rng` to underlying BrownianMotion
- Update dependencies, and support Python 3.8+

0.6.0 (2020-11-02)

- Removes zero args for dropping first sample vector value (breaking)
- Changes to diffusion process classes to align with common definitions (breaking)
- Refactor into processes and utils subpackages (breaking)
- Move base class checks into utils.validation and create abstract base classes for processes
- Provide RNG control and seeding functionality per instance and globally
- Add Dirichlet process
- Add generalized Diffusion process

0.5.0 (2020-09-22)

- Fixed a bug with missing drift when sampling Brownian motion at specific times (thanks to MichaelHogervorst)
- Fixed implementation of fractional Brownian motion (thanks to Antony Lee)
- Fixed a bug with Bernoulli process success probability

0.4.0 (2018-08-19)

- Added a MixedPoissonProcess (thanks to Gabinou)

0.3.0 (2018-07-22)

- Introduced breaking changes that move the t argument of all processes to the end of the __init__ signature
- Added support for inverse Gaussian process

0.2.0 (2018-07-11)

- Added support for colored noise processes (generalized power law, violet, blue, white, pink, red/Brownian)
- Added support for multifractional brownian motion
- Added more citations and bibliographical source page to docs

0.1.0 (2018-01-04)

- First release.
- Support for multiple continuous-time, discrete-time, diffusion, and noise processes.

CHAPTER

SIX

INDICES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

O

P

R

S

T

U

V

W