



# Timewave Documentation

*Release 0.6 [4 - Beta]*

**sonntagsgesicht, based on a fork of Deutsche Postbank [pbrisk**

**Wednesday, 18 September 2019**



---

## Contents

---

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Python library <i>timewave</i>	3
1.1.1	simulation engine	3
1.2	The State	3
1.3	The Producer	4
1.4	The Consumer	4
1.5	The Engine	4
1.6	main frame workflow	4
1.7	Development Version	5
1.8	Contributions	5
1.9	License	5
<b>2</b>	<b>Tutorial</b>	<b>7</b>
<b>3</b>	<b>API Documentation</b>	<b>9</b>
3.1	Timewave Engine	9
3.2	Timewave Producer	10
3.3	Timewave Consumer	11
3.4	Stochastic Process Simulation	14
3.5	Stochastic Process Definition	16
<b>4</b>	<b>Releases</b>	<b>25</b>
4.1	Release 0.6	25
4.2	Release 0.5	25
4.3	Release 0.4	25
4.4	Release 0.3	25
4.5	Release 0.2	25
4.6	Release 0.1	25
<b>5</b>	<b>Indices and tables</b>	<b>27</b>
	<b>Python Module Index</b>	<b>29</b>
	<b>Index</b>	<b>31</b>







### 1.1 Python library *timewave*

a stochastic process evolution simulation engine in python.

#### 1.1.1 simulation engine

timewave consists of four building blocks.

### 1.2 The State

which evolves over time during a simulation path. It is the nucleus or node which marks a point of time in a path.

## 1.3 The Producer

is the objects that provides states to the simulation and does the actual time evolution. Think of the producer building as the constructor of a stochastic process like a Brownian motion or, less mathematical, future stock prices or future rain intensities.

## 1.4 The Consumer

is an object that takes a state as a point in time provided by the producer and consumes it, i.e. does something with it - the actual calculation if you like.

## 1.5 The Engine

finally, which organizes the creation of states by the producer and the consumption by the consumer. The engine uses, if present, multiprocessing, i.e. takes full advantage of multi cpu frameworks. Therefore the engine splits the simulation into equal but distinct chunks of path for the number of workers (by default the number of cpu) and loops over the set of dedicated path in each worker. Each path is evolved by the producer in states which are at each point in time consumed directly by consumers. States are, due to limits of resources, not stored during the simulation. If you like to, use the storage consumer to save all states.

## 1.6 main frame workflow

setup simulation by

```
engine = Engine(Producer(), Consumer())
engine.run(range(20))
```

then run loop starts by

```
producer.initialize()
```

setup workers (by default by the number of cpu's) on each worker start loop by

```
producer/consumer.initialize_worker()
```

and invoke loop over paths and start again with

```
producer/consumer.initialize_path()
```

then do time evolution of a path

```
producer.evolve() / consumer.consume()
```

and finish with last consumer in consumer stack

```
consumer[-1].finalize_path()
```

and

```
consumer[-1].finalize_worker()
```

put results into queue and take them out by

```
consumer[-1].put() / get(result)
```



finish simulation (kind of reduce method)

```
consumer[-1].finalize()
```

before returning results from run.

## 1.7 Development Version

The latest development version can be installed directly from GitHub:

```
$ pip install --upgrade git+https://github.com/sonntagsgesicht/timewave.git
```

## 1.8 Contributions

Issues and [Pull Requests](#) are always welcome.

## 1.9 License

Code and documentation are available according to the Apache Software License (see [LICENSE](#)).



## CHAPTER 2

---

### Tutorial

---

setup simulation by

```
engine = Engine(Producer(), Consumer())  
engine.run(range(20))
```

then run loop starts by

```
producer.initialize()
```

setup workers (by default by the number of cpu's) on each worker start loop by

```
producer/consumer.initialize_worker()
```

and invoke loop over paths and start again with

```
producer/consumer.initialize_path()
```

then do time evolution of a path

```
producer.evolve() / consumer.consume()
```

and finish with last consumer in consumer stack

```
consumer[-1].finalize_path()
```

and

```
consumer[-1].finalize_worker()
```

put results into queue and take them out by

```
consumer[-1].put()/get(result)
```

finish simulation (kind of reduce method)

```
consumer[-1].finalize()
```

before returning results from run.



### 3.1 Timewave Engine

<i>Engine</i>	This class implements Monte Carlo engine
<i>State</i>	simulation state
<i>Producer</i>	abstract class implementing simple producer for a model between grid dates
<i>Consumer</i>	base class for simulation consumers

module containing simulation method related classes incl. multiprocessing support

**class** timewave.engine.**Producer** (*func=None, initial\_state=None*)

Bases: object

abstract class implementing simple producer for a model between grid dates

**initialize** (*grid, num\_of\_paths, seed*)

inits producer for a simulation run

**initialize\_worker** (*process\_num=None*)

inits producer for a simulation run on a single process

**initialize\_path** (*path\_num=None*)

inits producer for next path, i.e. sets current state to initial state

**evolve** (*new\_date*)

evolve to the new process state at the next date, i.e. do one step in the simulation

**Parameters** **new\_date** (*date*) – date of the new state

**Return State**

**class** timewave.engine.**State** (*value=0.0*)

Bases: object

simulation state

**class** timewave.engine.**Engine** (*producer=None, consumer=None*)

Bases: object

This class implements Monte Carlo engine

**run** (*grid=None, num\_of\_paths=2000, seed=0, num\_of\_workers=4, profiling=False*)  
implements simulation

#### Parameters

- **grid** (*list (date)*) – list of Monte Carlo grid dates
- **num\_of\_paths** (*int*) – number of Monte Carlo paths
- **seed** (*hashable*) – seed used for rnds initialisation (additional adjustment in place)
- **or None num\_of\_workers** (*int*) – number of parallel workers (default: `cpu_count()`), if `None` no parallel processing is used
- **profiling** (*bool*) – signal whether to use profiling, `True` means used, else not

**Return object** final consumer state

It returns a list of lists. The list contains per path a list produced by consumer at observation dates

**class** `timewave.engine.Consumer` (*func=None*)

Bases: `object`

base class for simulation consumers

initiatlizes consumer by providing a function :param func: consumer function with exact 1 argument which will consume the producer state. Default will return *state.value*

**initialize** (*grid=None, num\_of\_paths=None, seed=None*)

initialize consumer for simulation :param num\_of\_paths: number of path :type num\_of\_paths: int  
:param grid: list of grid point :type grid: list(date) :param seed: simulation seed :type seed: hashable

**initialize\_worker** (*process\_num=None*)

reinitialize consumer for process in multiprocessing

**initialize\_path** (*path\_num=None*)

initialize consumer for next path

**consume** (*state*)

consume new producer state

**finalize\_path** (*path\_num=None*)

finalize last path for consumer

**finalize\_worker** (*process\_num=None*)

finalize process for consumer

**finalize** ()

finalize simulation for consumer

**put** ()

to put state into multiprocessing.queue

**get** (*queue\_get*)

to get states from multiprocessing.queue

## 3.2 Timewave Producer

---

*DeterministicProducer*

---

*StringReaderProducer*

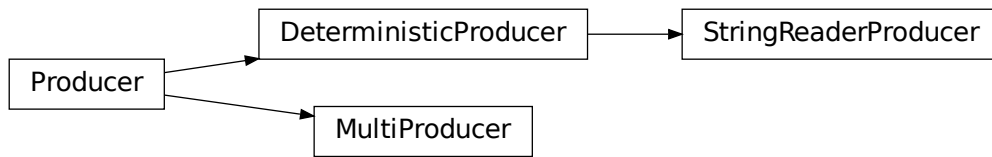
---

*MultiProducer*

---

initializer

---



module containing brownian motion model related classes

```

class timewave.producers.MultiProducer (*producers)
    Bases: timewave.engine.Producer

    initializer

        Parameters or produces (list (Producer)) – list of producers to be used one after
        another

    producers = None
        list of consumers to be used one after another

        Type list(Producer)

    initialize (grid, num_of_paths, seed)
        inits producer for a simulation run

    initialize_worker (process_num=None)
        inits producer for a simulation run on a single process

    initialize_path (path_num=None)
        inits producer for next path, i.e. sets current state to initial state

    evolve (new_date)
        evolve to the new process state at the next date, i.e. do one step in the simulation

        Parameters new_date (date) – date of the new state

        Return State

class timewave.producers.DeterministicProducer (sample_list, func=None, ini-
                                                tial_state=None)
    Bases: timewave.engine.Producer

    evolve (new_date)
        evolve to the new process state at the next date, i.e. do one step in the simulation

        Parameters new_date (date) – date of the new state

        Return State

class timewave.producers.StringReaderProducer (data_str, str_decoder=None)
    Bases: timewave.producers.DeterministicProducer
  
```

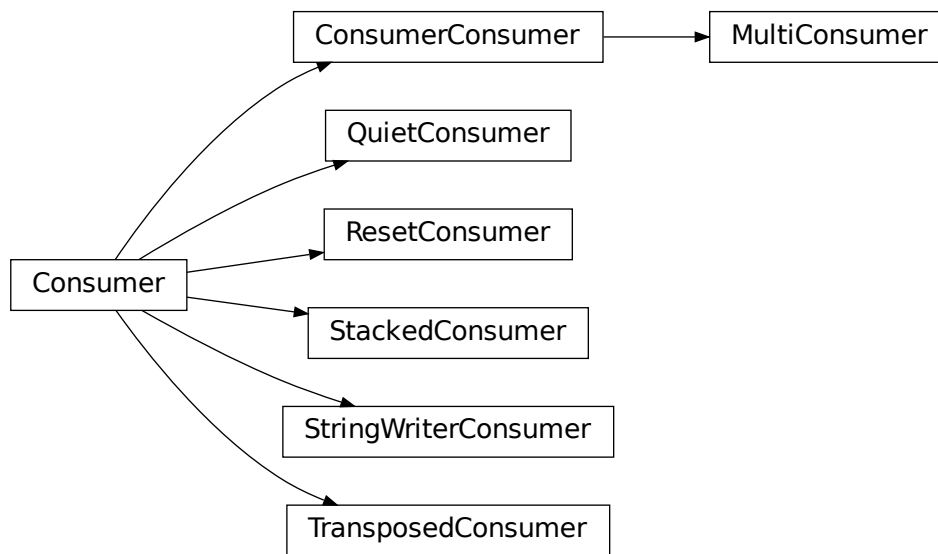
### 3.3 Timewave Consumer

<i>QuietConsumer</i>	QuietConsumer returns nothing, since QuietConsumer does simply not populate result in finalize_path()
<i>StringWriterConsumer</i>	
<i>StackedConsumer</i>	stacked version of consumer, i.e.

Continued on next page

Table 3 – continued from previous page

<i>ConsumerConsumer</i>	class implementing the consumer interface several consumers can be saved and are executed one after another only the result of the last consumer is returned (see <code>finalize_worker</code> )
<i>MultiConsumer</i>	initializer
<i>ResetConsumer</i>	FunctionConsumer that admits a reset function for each path
<i>TransposedConsumer</i>	TransposedConsumer returns sample distribution per grid point not per sample path



```
class timewave.consumers.QuietConsumer (func=None)
```

Bases: `timewave.engine.Consumer`

QuietConsumer returns nothing, since QuietConsumer does simply not populate result in `finalize_path()`

initiatlizes consumer by providing a function :param func: consumer function with exact 1 argument which will consume the producer state. Default will return `state.value`

```
finalize_path (path_num=None)
```

QuietConsumer does simply not populate result in `finalize_path()`

```
finalize ()
```

finalize simulation for consumer

```
class timewave.consumers.StringWriterConsumer (str_decoder=None)
```

Bases: `timewave.engine.Consumer`

```
finalize ()
```

finalize simulation for consumer

```
class timewave.consumers.ResetConsumer (fixing_func=None, reset_func=None)
```

Bases: `timewave.engine.Consumer`

FunctionConsumer that admits a reset function for each path



```

initialize_path (path_num=None)
    initialize consumer for next path

finalize ()
    finalize simulation for consumer

class timewave.consumers.StackedConsumer (*consumers)
    Bases: timewave.engine.Consumer

    stacked version of consumer, i.e. a following consumer is populated with out state of the preceding one

    initialize (num_of_paths=None, grid=None, seed=None)
        initialize StackedConsumer

    initialize_path (path_num=None)
        make the consumer_state ready for the next MC path

        Parameters path_num (int) –

    consume (state)
        consume new producer state

    finalize_path (path_num=None)
        finalize path and populate result for ConsumerConsumer

    finalize ()
        finalize for ConsumerConsumer

    put ()
        to put state into multiprocessing.queue

    get (queue_get)
        to get states from multiprocessing.queue

class timewave.consumers.ConsumerConsumer (*consumers)
    Bases: timewave.engine.Consumer

    class implementing the consumer interface several consumers can be saved and are executed one after
    another only the result of the last consumer is returned (see finalize_worker)

    initializer

        Parameters list (Consumer) –

    initial_state = None
        list of consumers to be used one after another

        Type list(Consumer)

    initialize (grid=None, num_of_paths=None, seed=None)
        initialize ConsumerConsumer

    initialize_path (path_num=None)
        make the consumer_state ready for the next MC path

        Parameters path_num (int) –

    consume (state)
        returns pair containing the result of consumption and consumer state the returned state is equal to the
        state.get_short_rate() the returned consume state is None

        Parameters state (State) – specific process state

        Return object the new consumer state

    finalize_path (path_num=None)
        finalize path and populate result for ConsumerConsumer

    finalize ()
        finalize for ConsumerConsumer

```

**get** (*queue\_get*)

get to given consumer states. This function is used for merging of results of parallelized MC. The first state is used for merging in place. The states must be disjoint.

**Parameters** *queue\_get* (*object*) – second consumer state

**class** `timewave.consumers.MultiConsumer` (*\*consumers*)

Bases: `timewave.consumers.ConsumerConsumer`

initializer

**Parameters** *list (Consumer)* –

**consume** (*state*)

returns pair containing the result of consumption and consumer state the returned state is equal to the state.get\_short\_rate() the returned consume state is None

**Parameters** *state (State)* – specific process state

**Return object** the new consumer state

**class** `timewave.consumers.TransposedConsumer` (*func=None*)

Bases: `timewave.engine.Consumer`

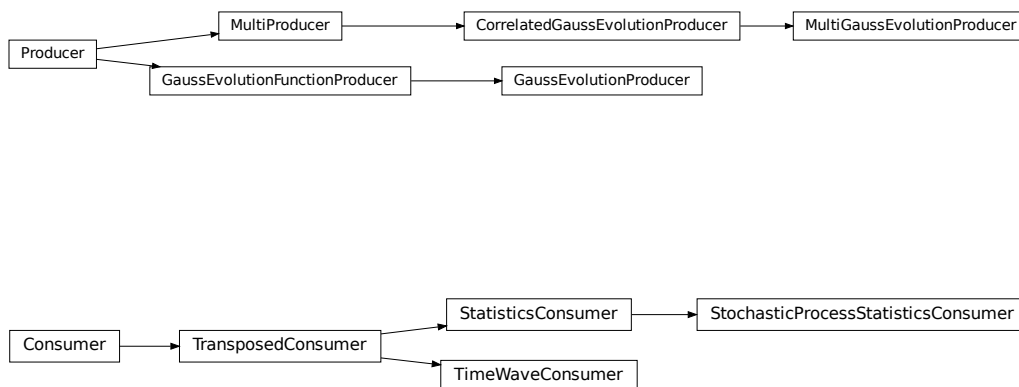
TransposedConsumer returns sample distribution per grid point not per sample path

initiatlizes consumer by providing a function :param func: consumer function with exact 1 argument which will consume the producer state. Default will return *state.value*

**finalize** ()

finalize for PathConsumer

## 3.4 Stochastic Process Simulation



module containing stochastic process model producer

**class** `timewave.stochasticproducer.GaussEvolutionFunctionProducer` (*func=None*,  
*ini-*  
*tial\_state=None*,  
*length=None*)

Bases: `timewave.engine.Producer`

class implementing general Gauss process between grid dates

### Parameters

- **func** (*callable*) – evolve function, e.g. `lambda x, s, e, q: x + sqrt(e - s) * q` by default with *x* current state value, *s* current point in time, i.e. start point of next evolution step, *e* next point in time, i.e. end point of evolution step, *q* standard normal random number to do step
- **initial\_state** – initial state (value) of evolution,
- **or None length** (*int*) – length of *q* as a list of Gauss random numbers, if *None* or *0* the evolution function *func* will be invoked with *q* not as a list but a float random number.

class implementing general Gauss process between grid dates and provides state to any evolve style function *foo(x, s, e, q)* with *x* last state, *s* last state time, *e* current point in time and *q* current Gauss process state

**evolve** (*new\_date*)

evolve to the new process state at the next date

**Parameters** *new\_date* (*date*) – date or point in time of the new state

### Return State

**class** `timewave.stochasticproducer.GaussEvolutionProducer` (*process*)

Bases: `timewave.stochasticproducer.GaussEvolutionFunctionProducer`

producer to bring diffusion process to life

**Parameters** *process* (`StochasticProcess`) – diffusion process to evolve

**class** `timewave.stochasticproducer.CorrelatedGaussEvolutionProducer` (*producers*,  
*cor-*  
*rela-*  
*tion=None*,  
*diffu-*  
*sion\_driver=None*)

Bases: `timewave.producers.MultiProducer`

class implementing general correlated Gauss process between grid dates

### Parameters

- **producers** (*list* (`GaussEvolutionProducer`)) – list of producers to evolve
- **or dict** ((`StochasticProcess`, `StochasticProcess`)  
(*list* (*list* (*float*))) – float) or *None* correlation: correlation matrix of underlying multivariate Gauss process of diffusion drivers. If *dict* keys must be pairs of diffusion drivers, diagonal and zero entries can be omitted. If not give, all drivers evolve independently.
- **or None diffusion\_driver** (*list* (`StochasticProcess`)) – list of diffusion drivers indexing the correlation matrix. If not given and *correlation* is not an `IndexMatrix`, e.g. comes already with list of drivers, it is assumed that each process producer has different drivers and the correlation is order in the same way.

**evolve** (*new\_date*)

evolve to the new process state at the next date

**Parameters** *new\_date* (*date*) – date or point in time of the new state

### Return State

**class** `timewave.stochasticproducer.MultiGaussEvolutionProducer` (*process\_list*,  
*correla-*  
*tion=None*,  
*diffu-*  
*sion\_driver=None*)

Bases: `timewave.stochasticproducer.CorrelatedGaussEvolutionProducer`

class implementing multi variant GaussEvolutionProducer

```
class timewave.stochasticconsumer.StatisticsConsumer (func=None, statistics=None, **kwargs)
```

Bases: *timewave.consumers.TransposedConsumer*

run basic statistics on storage consumer result per time slice

```
finalize ()  
    finalize for StatisticsConsumer
```

```
class timewave.stochasticconsumer.StochasticProcessStatisticsConsumer (func=None, statistics=None, **kwargs)
```

Bases: *timewave.stochasticconsumer.StatisticsConsumer*

run basic statistics on storage consumer result as a stochastic process

```
finalize ()  
    finalize for StochasticProcessStatisticsConsumer
```

```
class timewave.stochasticconsumer.TimeWaveConsumer (func=None)
```

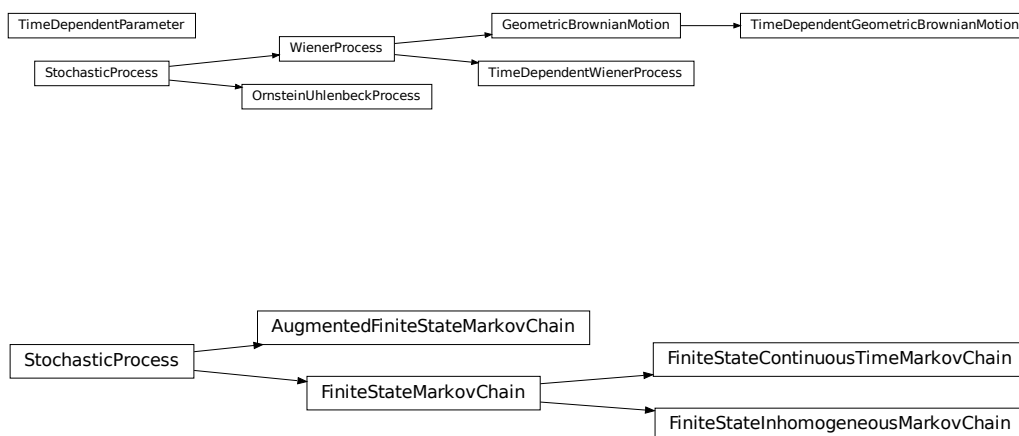
Bases: *timewave.consumers.TransposedConsumer*

initiatlizes consumer by providing a function :param func: consumer function with exact 1 argument which will consume the producer state. Default will return *state.value*

```
finalize ()  
    finalize for PathConsumer
```

## 3.5 Stochastic Process Definition

<i>stochasticprocess.base.StochasticProcess</i>	base class for stochastic process $X$ , e.g. Wiener process $W$ or Markov chain $M$
<i>stochasticprocess.base.MultivariateStochasticProcess</i>	base class for stochastic process $X$ , e.g. Wiener process $W$ or Markov chain $M$



---

```

class timewave.stochasticprocess.base.StochasticProcess (start)
    Bases: object
        base class for stochastic process  $X$ , e.g. Wiener process  $W$  or Markov chain  $M$ 

        Parameters start – initial state  $X_0$ 

    classmethod random ()
        initializes stochastic process with some randomly generated parameters

        Return type StochasticProcess

    diffusion_driver
        diffusion driver are the underlying  $dW$  of each process  $X$  in a SDE like  $dX = m dt + s dW$ 

        Return list(StochasticProcess)

    evolve (x, s, e, q)
        evolves process state  $x$  from  $s$  to  $e$  in time depending of standard normal random variable  $q$ 

        Parameters
            • x (object) – current state value, i.e. value before evolution step
            • s (float) – current point in time, i.e. start point of next evolution step
            • e (float) – next point in time, i.e. end point of evolution step
            • q (float) – standard normal random number to do step

        Returns next state value, i.e. value after evolution step

        Return type object

    mean (t)
        expected value of time  $t$  increment

        Parameters t (float) –

        Return type float

        Returns expected value of time  $t$  increment

    median (t)

    variance (t)
        second central moment of time  $t$  increment

        Parameters t (float) –

        Return type float

        Returns variance, i.e. second central moment of time  $t$  increment

    stdev (t)

    skewness (t)

    kurtosis (t)

class timewave.stochasticprocess.base.MultivariateStochasticProcess (start)
    Bases: timewave.stochasticprocess.base.StochasticProcess
        base class for stochastic process  $X$ , e.g. Wiener process  $W$  or Markov chain  $M$ 

        Parameters start – initial state  $X_0$ 

class timewave.stochasticprocess.gauss.WienerProcess (mu=0.0, sigma=1.0,
                                                         start=0.0)
    Bases: timewave.stochasticprocess.base.StochasticProcess
        class implementing general Gauss process between grid dates

```

**evolve** (*x*, *s*, *e*, *q*)

evolves process state *x* from *s* to *e* in time depending of standard normal random variable *q*

**Parameters**

- **x** (*object*) – current state value, i.e. value before evolution step
- **s** (*float*) – current point in time, i.e. start point of next evolution step
- **e** (*float*) – next point in time, i.e. end point of evolution step
- **q** (*float*) – standard normal random number to do step

**Returns** next state value, i.e. value after evolution step

**Return type** object

**mean** (*t*)

expected value of time *t* increment

**Parameters** **t** (*float*) –

**Return type** float

**Returns** expected value of time *t* increment

**median** (*t*)

**variance** (*t*)

second central moment of time *t* increment

**Parameters** **t** (*float*) –

**Return type** float

**Returns** variance, i.e. second central moment of time *t* increment

**class** timewave.stochasticprocess.gauss.**OrnsteinUhlenbeckProcess** (*theta=0.1*,  
*mu=0.1*,  
*sigma=0.1*,  
*start=0.0*)

Bases: *timewave.stochasticprocess.base.StochasticProcess*

class implementing Ornstein Uhlenbeck process

**Parameters**

- **theta** (*float*) – mean reversion speed
- **mu** (*float*) – drift
- **sigma** (*float*) – diffusion
- **start** (*float*) – initial value

$$dx_t = \theta(\mu - x_t)dt + \sigma dW_t, x_0 = a$$

**evolve** (*x*, *s*, *e*, *q*)

evolves process state *x* from *s* to *e* in time depending of standard normal random variable *q*

**Parameters**

- **x** (*object*) – current state value, i.e. value before evolution step
- **s** (*float*) – current point in time, i.e. start point of next evolution step
- **e** (*float*) – next point in time, i.e. end point of evolution step
- **q** (*float*) – standard normal random number to do step

**Returns** next state value, i.e. value after evolution step

**Return type** object

**mean** (*t*)  
 expected value of time *t* increment  
**Parameters** *t* (*float*) –  
**Return type** *float*  
**Returns** expected value of time *t* increment

**variance** (*t*)  
 second central moment of time *t* increment  
**Parameters** *t* (*float*) –  
**Return type** *float*  
**Returns** variance, i.e. second central moment of time *t* increment

**class** timewave.stochasticprocess.gauss.GeometricBrownianMotion (*mu=0.0*,  
*sigma=1.0*,  
*start=1.0*)

Bases: *timewave.stochasticprocess.gauss.WienerProcess*

class implementing general Gauss process between grid dates

**evolve** (*x*, *s*, *e*, *q*)  
 evolves process state *x* from *s* to *e* in time depending of standard normal random variable *q*

**Parameters**

- *x* (*object*) – current state value, i.e. value before evolution step
- *s* (*float*) – current point in time, i.e. start point of next evolution step
- *e* (*float*) – next point in time, i.e. end point of evolution step
- *q* (*float*) – standard normal random number to do step

**Returns** next state value, i.e. value after evolution step

**Return type** *object*

**mean** (*t*)  
 expected value of time *t* increment  
**Parameters** *t* (*float*) –  
**Return type** *float*  
**Returns** expected value of time *t* increment

**median** (*t*)

**variance** (*t*)  
 second central moment of time *t* increment  
**Parameters** *t* (*float*) –  
**Return type** *float*  
**Returns** variance, i.e. second central moment of time *t* increment

**skewness** (*t*)

**kurtosis** (*t*)

**class** timewave.stochasticprocess.gauss.TimeDependentParameter (*parameter=0.0*,  
*time=1.0*)

Bases: *object*

**integrate** (*s*, *e*)

```
class timewave.stochasticprocess.gauss.TimeDependentWienerProcess (mu=0.0,  
                                                                sigma=1.0,  
                                                                time=1.0,  
                                                                start=0.0)
```

Bases: *timewave.stochasticprocess.gauss.WienerProcess*

class implementing a Gauss process with time depending drift and diffusion

```
class timewave.stochasticprocess.gauss.TimeDependentGeometricBrownianMotion (mu=0.0,  
                                                                sigma=1.0,  
                                                                time=1.0,  
                                                                start=1.0)
```

Bases: *timewave.stochasticprocess.gauss.GeometricBrownianMotion*

```
class timewave.stochasticprocess.markovchain.FiniteStateMarkovChain (transition=None,  
                                                                r_squared=1.0,  
                                                                start=None)
```

Bases: *timewave.stochasticprocess.base.StochasticProcess*

#### Parameters

- **transition** (*list*) – stochastic matrix of transition probabilities, i.e. `np.ndarray` with shape=2 and sum of each line equal to 1 (optional) default: identity matrix
- **r\_squared** (*float*) – square of systematic correlation in factor simulation (optional) default: 1.
- **start** (*list*) – initial state distribution, i.e. `np.ndarray` with shape=1 or list, adding up to 1, (optional) default: unique distribution

**transition**

**r\_squared**

**classmethod random** (*d=5*)

initializes stochastic process with some randomly generated parameters

**Return type** *StochasticProcess*

**evolve** (*x, s, e, q*)

evolves process state *x* from *s* to *e* in time depending of standard normal random variable *q*

#### Parameters

- **x** (*object*) – current state value, i.e. value before evolution step
- **s** (*float*) – current point in time, i.e. start point of next evolution step
- **e** (*float*) – next point in time, i.e. end point of evolution step
- **q** (*float*) – standard normal random number to do step

**Returns** next state value, i.e. value after evolution step

**Return type** object

**mean** (*t*)

expected value of time *t* increment

**Parameters** **t** (*float*) –

**Return type** float

**Returns** expected value of time *t* increment

**variance** (*t*)

second central moment of time *t* increment

**Parameters** **t** (*float*) –

**Return type** float



**Returns** variance, i.e. second central moment of time  $t$  increment

**covariance** ( $t$ )

**class** timewave.stochasticprocess.markovchain.FiniteStateContinuousTimeMarkovChain (*transition*  
*r\_square*  
*start=None*)

Bases: *timewave.stochasticprocess.markovchain.FiniteStateMarkovChain*

**class** timewave.stochasticprocess.markovchain.FiniteStateInhomogeneousMarkovChain (*transition*  
*r\_square*  
*start=None*)

Bases: *timewave.stochasticprocess.markovchain.FiniteStateMarkovChain*

**classmethod** random ( $d=5, l=3$ )

initializes stochastic process with some randomly generated parameters

**Return type** *StochasticProcess*

**class** timewave.stochasticprocess.markovchain.AugmentedFiniteStateMarkovChain (*underlying*,  
*aug-*  
*men-*  
*ta-*  
*tion=None*)

Bases: *timewave.stochasticprocess.base.StochasticProcess*

#### Parameters

- **underlying** (*FiniteStateMarkovChain*) – underlying Markov chain stochastic process
- **augmentation** (*callable*) – function  $f : S \rightarrow \mathbb{R}$  defined on single states to weight augmentation (aggregate) of state distributions (optional) default:  $f = id$  Augmentation argument can be list or tuple, too. In this case `__getitem__` is called.

**classmethod** random ( $d=5, augmentation=None$ )

initializes stochastic process with some randomly generated parameters

**Return type** *StochasticProcess*

**diffusion\_driver**

diffusion driver are the underlying  $dW$  of each process  $X$  in a SDE like  $dX = m dt + s dW$

**Return list**(*StochasticProcess*)

**start**

**evolve** ( $x, s, e, q$ )

evolves process state  $x$  from  $s$  to  $e$  in time depending of standard normal random variable  $q$

#### Parameters

- **x** (*object*) – current state value, i.e. value before evolution step
- **s** (*float*) – current point in time, i.e. start point of next evolution step
- **e** (*float*) – next point in time, i.e. end point of evolution step
- **q** (*float*) – standard normal random number to do step

**Returns** next state value, i.e. value after evolution step

**Return type** object

**eval** ( $s$ )

**mean** ( $t$ )

expected value of time  $t$  increment

**Parameters** **t** (*float*) –

**Return type** float

**Returns** expected value of time  $t$  increment

**variance** ( $t$ )

second central moment of time  $t$  increment

**Parameters**  $t$  (*float*) –

**Return type** float

**Returns** variance, i.e. second central moment of time  $t$  increment

**class** timewave.stochasticprocess.multifactor.**SABR** ( $\alpha=0.1$ ,  $\beta=0.2$ ,  $\nu=0.3$ ,  
 $\rho=-0.2$ ,  $\text{start}=0.05$ )

Bases: *timewave.stochasticprocess.base.MultivariateStochasticProcess*

class implementing the Hagan et al SABR model

**evolve** ( $x, s, e, q$ )

evolves process state  $x$  from  $s$  to  $e$  in time depending of standard normal random variable  $q$

**Parameters**

- $\mathbf{x}$  (*object*) – current state value, i.e. value before evolution step
- $\mathbf{s}$  (*float*) – current point in time, i.e. start point of next evolution step
- $\mathbf{e}$  (*float*) – next point in time, i.e. end point of evolution step
- $\mathbf{q}$  (*float*) – standard normal random number to do step

**Returns** next state value, i.e. value after evolution step

**Return type** object

**mean** ( $t$ )

expected value of time  $t$  increment

**Parameters**  $t$  (*float*) –

**Return type** float

**Returns** expected value of time  $t$  increment

**variance** ( $t$ )

second central moment of time  $t$  increment

**Parameters**  $t$  (*float*) –

**Return type** float

**Returns** variance, i.e. second central moment of time  $t$  increment

**class** timewave.stochasticprocess.multifactor.**MultiGauss** ( $\mu=[0.0]$ ,  $\text{co-}$   
 $\text{var}=[[1.0]]$ ,  
 $\text{start}=[0.0]$ )

Bases: *timewave.stochasticprocess.base.MultivariateStochasticProcess*

class implementing multi dimensional brownian motion

**evolve** ( $x, s, e, q$ )

evolves process state  $x$  from  $s$  to  $e$  in time depending of standard normal random variable  $q$

**Parameters**

- $\mathbf{x}$  (*object*) – current state value, i.e. value before evolution step
- $\mathbf{s}$  (*float*) – current point in time, i.e. start point of next evolution step
- $\mathbf{e}$  (*float*) – next point in time, i.e. end point of evolution step
- $\mathbf{q}$  (*float*) – standard normal random number to do step

**Returns** next state value, i.e. value after evolution step

**Return type** object

**mean** ( $t$ )

expected value of time  $t$  increment

**Parameters**  $\mathfrak{t}$  (*float*) –

**Return type** float

**Returns** expected value of time  $t$  increment

**variance** ( $t$ )

second central moment of time  $t$  increment

**Parameters**  $\mathfrak{t}$  (*float*) –

**Return type** float

**Returns** variance, i.e. second central moment of time  $t$  increment



## CHAPTER 4

---

### Releases

---

These changes are listed in decreasing version number order.

#### **4.1 Release 0.6**

Release date was Wednesday, 18 September 2019

#### **4.2 Release 0.5**

Release date was July 14th, 2018

#### **4.3 Release 0.4**

Release date was July 7th, 2017

#### **4.4 Release 0.3**

Release date was April 27th, 2017

#### **4.5 Release 0.2**

Release date was April 2nd, 2017

#### **4.6 Release 0.1**

Release date was April 2nd, 2016



## CHAPTER 5

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`





### t

- `timewave`, [7](#)
- `timewave.consumers`, [12](#)
- `timewave.engine`, [9](#)
- `timewave.producers`, [11](#)
- `timewave.stochasticconsumer`, [16](#)
- `timewave.stochasticprocess.base`, [16](#)
- `timewave.stochasticprocess.gauss`, [17](#)
- `timewave.stochasticprocess.markovchain`,  
[20](#)
- `timewave.stochasticprocess.multifactor`,  
[22](#)
- `timewave.stochasticproducer`, [14](#)



## A

AugmentedFiniteStateMarkovChain (class in timewave.stochasticprocess.markovchain), 21

## C

consume() (timewave.consumers.ConsumerConsumer method), 13

consume() (timewave.consumers.MultiConsumer method), 14

consume() (timewave.consumers.StackedConsumer method), 13

consume() (timewave.engine.Consumer method), 10

Consumer (class in timewave.engine), 10

ConsumerConsumer (class in timewave.consumers), 13

CorrelatedGaussEvolutionProducer (class in timewave.stochasticproducer), 15

covariance() (timewave.stochasticprocess.markovchain.FiniteStateMarkovChain method), 21

## D

DeterministicProducer (class in timewave.producers), 11

diffusion\_driver (timewave.stochasticprocess.base.StochasticProcess attribute), 17

diffusion\_driver (timewave.stochasticprocess.markovchain.AugmentedFiniteStateMarkovChain attribute), 21

## E

Engine (class in timewave.engine), 9

eval() (timewave.stochasticprocess.markovchain.AugmentedFiniteStateMarkovChain method), 21

evolve() (timewave.engine.Producer method), 9

evolve() (timewave.producers.DeterministicProducer method), 11

evolve() (timewave.producers.MultiProducer method), 11

evolve() (timewave.stochasticprocess.base.StochasticProcess method), 17

evolve() (timewave.stochasticprocess.gauss.GeometricBrownianMotion method), 19

evolve() (timewave.stochasticprocess.gauss.OrnsteinUhlenbeckProcess method), 18

evolve() (timewave.stochasticprocess.gauss.WienerProcess method), 17

evolve() (timewave.stochasticprocess.markovchain.AugmentedFiniteStateMarkovChain method), 21

evolve() (timewave.stochasticprocess.markovchain.FiniteStateMarkovChain method), 20

evolve() (timewave.stochasticprocess.multifactor.MultiGauss method), 22

evolve() (timewave.stochasticprocess.multifactor.SABR method), 22

evolve() (timewave.stochasticproducer.CorrelatedGaussEvolutionProducer method), 15

evolve() (timewave.stochasticproducer.GaussEvolutionFunctionProducer method), 15

## F

finalize() (timewave.consumers.ConsumerConsumer method), 13

finalize() (timewave.consumers.QuietConsumer method), 12

finalize() (timewave.consumers.ResetConsumer method), 13

finalize() (timewave.consumers.StackedConsumer method), 13

finalize() (timewave.consumers.StringWriterConsumer method), 12

finalize() (timewave.consumers.TransposedConsumer method), 14

finalize() (timewave.engine.Consumer method), 10

finalize() (timewave.stochasticconsumer.StatisticsConsumer method), 16

finalize() (timewave.stochasticconsumer.StochasticProcessStatisticsConsumer method), 16

[finalize\(\)](#) (time-wave.stochasticconsumer.TimeWaveConsumer method), 16  
[finalize\\_path\(\)](#) (time-wave.consumers.ConsumerConsumer method), 13  
[finalize\\_path\(\)](#) (time-wave.consumers.QuietConsumer method), 12  
[finalize\\_path\(\)](#) (time-wave.consumers.StackedConsumer method), 13  
[finalize\\_path\(\)](#) (timewave.engine.Consumer method), 10  
[finalize\\_worker\(\)](#) (timewave.engine.Consumer method), 10  
[FiniteStateContinuousTimeMarkovChain](#) (class in time-wave.stochasticprocess.markovchain), 21  
[FiniteStateInhomogeneousMarkovChain](#) (class in time-wave.stochasticprocess.markovchain), 21  
[FiniteStateMarkovChain](#) (class in time-wave.stochasticprocess.markovchain), 20

## G

[GaussEvolutionFunctionProducer](#) (class in timewave.stochasticproducer), 14  
[GaussEvolutionProducer](#) (class in time-wave.stochasticproducer), 15  
[GeometricBrownianMotion](#) (class in time-wave.stochasticprocess.gauss), 19  
[get\(\)](#) (timewave.consumers.ConsumerConsumer method), 13  
[get\(\)](#) (timewave.consumers.StackedConsumer method), 13  
[get\(\)](#) (timewave.engine.Consumer method), 10

## I

[initial\\_state](#) (time-wave.consumers.ConsumerConsumer attribute), 13  
[initialize\(\)](#) (time-wave.consumers.ConsumerConsumer method), 13  
[initialize\(\)](#) (time-wave.consumers.StackedConsumer method), 13  
[initialize\(\)](#) (timewave.engine.Consumer method), 10  
[initialize\(\)](#) (timewave.engine.Producer method), 9  
[initialize\(\)](#) (timewave.producers.MultiProducer method), 11  
[initialize\\_path\(\)](#) (time-wave.consumers.ConsumerConsumer method), 13  
[initialize\\_path\(\)](#) (time-wave.consumers.ResetConsumer method), 12  
[initialize\\_path\(\)](#) (time-wave.consumers.StackedConsumer method), 13  
[initialize\\_path\(\)](#) (timewave.engine.Consumer method), 10  
[initialize\\_path\(\)](#) (timewave.engine.Producer method), 9  
[initialize\\_path\(\)](#) (time-wave.producers.MultiProducer method), 11  
[initialize\\_worker\(\)](#) (time-wave.engine.Consumer method), 10  
[initialize\\_worker\(\)](#) (time-wave.engine.Producer method), 9  
[initialize\\_worker\(\)](#) (time-wave.producers.MultiProducer method), 11  
[integrate\(\)](#) (time-wave.stochasticprocess.gauss.TimeDependentParameter method), 19

## K

[kurtosis\(\)](#) (time-wave.stochasticprocess.base.StochasticProcess method), 17  
[kurtosis\(\)](#) (time-wave.stochasticprocess.gauss.GeometricBrownianMotion method), 19

## M

[mean\(\)](#) (timewave.stochasticprocess.base.StochasticProcess method), 17  
[mean\(\)](#) (timewave.stochasticprocess.gauss.GeometricBrownianMotion method), 19  
[mean\(\)](#) (timewave.stochasticprocess.gauss.OrnsteinUhlenbeckProcess method), 19  
[mean\(\)](#) (timewave.stochasticprocess.gauss.WienerProcess method), 18  
[mean\(\)](#) (timewave.stochasticprocess.markovchain.AugmentedFiniteState method), 21  
[mean\(\)](#) (timewave.stochasticprocess.markovchain.FiniteStateMarkovChain method), 20  
[mean\(\)](#) (timewave.stochasticprocess.multifactor.MultiGauss method), 23  
[mean\(\)](#) (timewave.stochasticprocess.multifactor.SABR method), 22  
[median\(\)](#) (timewave.stochasticprocess.base.StochasticProcess method), 17  
[median\(\)](#) (timewave.stochasticprocess.gauss.GeometricBrownianMotion method), 19  
[median\(\)](#) (timewave.stochasticprocess.gauss.WienerProcess method), 18  
[MultiConsumer](#) (class in timewave.consumers), 14

MultiGauss (class in timewave.stochasticprocess.multifactor), 22  
 MultiGaussEvolutionProducer (class in timewave.stochasticproducer), 15  
 MultiProducer (class in timewave.producers), 11  
 MultivariateStochasticProcess (class in timewave.stochasticprocess.base), 17

## O

OrnsteinUhlenbeckProcess (class in timewave.stochasticprocess.gauss), 18

## P

Producer (class in timewave.engine), 9  
 producers (timewave.producers.MultiProducer attribute), 11  
 put () (timewave.consumers.StackedConsumer method), 13  
 put () (timewave.engine.Consumer method), 10

## Q

QuietConsumer (class in timewave.consumers), 12

## R

r\_squared (timewave.stochasticprocess.markovchain.FiniteStateMarkovChain attribute), 20  
 random () (timewave.stochasticprocess.base.StochasticProcess class method), 17  
 random () (timewave.stochasticprocess.markovchain.AugmentedFiniteStateMarkovChain class method), 21  
 random () (timewave.stochasticprocess.markovchain.FiniteStateInhomogeneousMarkovChain class method), 21  
 random () (timewave.stochasticprocess.markovchain.FiniteStateMarkovChain class method), 20  
 ResetConsumer (class in timewave.consumers), 12  
 run () (timewave.engine.Engine method), 9

## S

SABR (class in timewave.stochasticprocess.multifactor), 22  
 skewness () (timewave.stochasticprocess.base.StochasticProcess method), 17  
 skewness () (timewave.stochasticprocess.gauss.GeometricBrownianMotion method), 19  
 StackedConsumer (class in timewave.consumers), 13  
 start (timewave.stochasticprocess.markovchain.AugmentedFiniteStateMarkovChain attribute), 21  
 State (class in timewave.engine), 9  
 StatisticsConsumer (class in timewave.stochasticconsumer), 16  
 stdev () (timewave.stochasticprocess.base.StochasticProcess method), 17  
 StochasticProcess (class in timewave.stochasticprocess.base), 16

StochasticProcessStatisticsConsumer (class in timewave.stochasticconsumer), 16  
 StringReaderProducer (class in timewave.producers), 11  
 StringWriterConsumer (class in timewave.consumers), 12

## T

TimeDependentGeometricBrownianMotion (class in timewave.stochasticprocess.gauss), 20  
 TimeDependentParameter (class in timewave.stochasticprocess.gauss), 19  
 TimeDependentWienerProcess (class in timewave.stochasticprocess.gauss), 19  
 timewave (module), 7  
 timewave.consumers (module), 12  
 timewave.engine (module), 9  
 timewave.producers (module), 11  
 timewave.stochasticconsumer (module), 16  
 timewave.stochasticprocess.base (module), 16  
 timewave.stochasticprocess.gauss (module), 17  
 timewave.stochasticprocess.markovchain (module), 20  
 timewave.stochasticprocess.multifactor (module), 22  
 timewave.stochasticproducer (module), 14  
 timewaveconsumer (class in timewave.stochasticconsumer), 16  
 transition (timewave.stochasticprocess.markovchain.FiniteStateMarkovChain attribute), 20  
 TransposedConsumer (class in timewave.consumers), 14

## V

variance () (timewave.stochasticprocess.base.StochasticProcess method), 17  
 variance () (timewave.stochasticprocess.gauss.GeometricBrownianMotion method), 19  
 variance () (timewave.stochasticprocess.gauss.OrnsteinUhlenbeckProcess method), 19  
 variance () (timewave.stochasticprocess.gauss.WienerProcess method), 18  
 variance () (timewave.stochasticprocess.markovchain.AugmentedFiniteStateMarkovChain method), 22  
 variance () (timewave.stochasticprocess.markovchain.FiniteStateMarkovChain method), 20  
 variance () (timewave.stochasticprocess.multifactor.MultiGauss

*method*), [23](#)  
variance() (time-  
wave.stochasticprocess.multifactor.SABR  
*method*), [22](#)

## W

WienerProcess (class in time-  
wave.stochasticprocess.gauss), [17](#)