proba_num

Generated by Doxygen 1.8.14

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

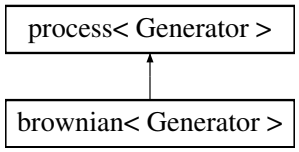
Class Documentation

3.1 brownian< Generator > Class Template Reference

A class to discretize a Brownian Motion with a classical Euler scheme.

`#include <brownian.h>`

Inheritance diagram for brownian< Generator >:



Public Member Functions

- brownian ()

  Constructors & destructors.
- **brownian** (double vol)
- std::vector< double > next_step (Generator &gen, std::vector< double > state, double t)

  Next step.

Additional Inherited Members

3.1.1 Detailed Description

template<typename Generator>
class brownian< Generator >

A class to discretize a Brownian Motion with a classical Euler scheme.

Mainly used for testing.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 brownian()

```
template<typename Generator >
brownian< Generator >::brownian ( )
```

Constructors & destructors.

Functions for the brownian class.

The documentation for this class was generated from the following file:

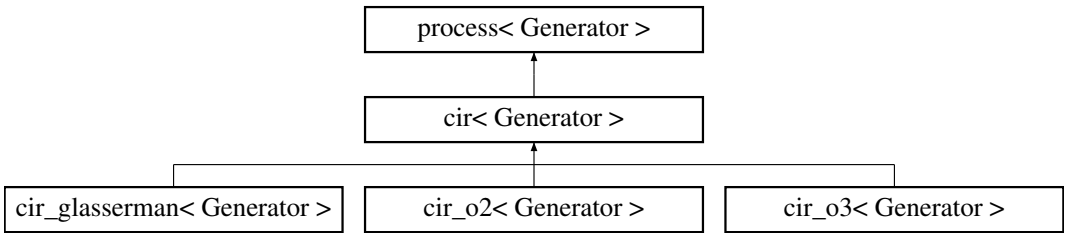
- brownian.h

3.2 cir< Generator > Class Template Reference

Generation of a class with a CIR process.

```
#include <cir.h>
```

Inheritance diagram for cir< Generator >:



Public Member Functions

- cir ()
  Constructor.
- cir (double state_0, double k, double a, double s)
  Constructor.
- ~cir ()
  Destructor.
- virtual std::vector< double > next_step (Generator &gen, std::vector< double > state, double t)=0
  Pure virtual function. Will be implemented in the different schemes.
- double get_a ()
  Getter.
- double get_sigma ()
  Getter.
- void set_a (double a)
  Setter.
- void set_sigma (double s)
  Setter.

Protected Attributes

- double a
- double sigma
- double k

3.2.1 Detailed Description

template<typename Generator>
class cir< Generator >

Generation of a class with a CIR process.

The class cir implements the class process. The CIR follows the dynamics dX = (a - kX) dt + sigma sqrt(X) dW with X(0) = x_0

3.2.2 Member Data Documentation

3.2.2.1 a

```
template<typename Generator >
double cir< Generator >::a  [protected]
```

Target value for CIR

3.2.2.2 k

```
template<typename Generator >
double cir< Generator >::k  [protected]
```

Mean reverting speed

3.2.2.3 sigma

```
template<typename Generator >
double cir< Generator >::sigma  [protected]
```

Volatility of volatility

The documentation for this class was generated from the following file:

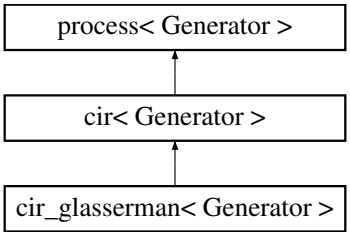
- cir.h

3.3 cir_glasserman< Generator > Class Template Reference

An exact scheme for the CIR process.

`#include <cir.h>`

Inheritance diagram for cir_glasserman< Generator >:



Public Member Functions

- cir_glasserman ()
  Constructor.
- cir_glasserman (double state_0, double k, double a, double s)
  Constructor.
- ~cir_glasserman ()
  Destructor.
- std::vector< double > next_step (Generator &gen, std::vector< double > state, double t)
  Pure virtual function. Will be implemented in the different schemes.

Protected Attributes

- std::poisson_distribution< int > n_poisson
  Getters and Setters.
- std::chi_squared_distribution< double > x_chi
- std::normal_distribution< double > z_norm
- double d

3.3.1 Detailed Description

template<typename Generator>
class cir_glasserman< Generator >

An exact scheme for the CIR process.

This class implements an exact scheme of discretization of the CIR process. It exploits the knowledge of the transition law of this process. It can be found in a book of Paul Glasserman.

3.3.2 Member Data Documentation

3.3.2.1 d

```
template<typename Generator >
double cir_glasserman< Generator >::d  [protected]
```

An auxiliary variable

3.3.2.2 n_poisson

```
template<typename Generator >
std::poisson_distribution<int> cir_glasserman< Generator >::n_poisson  [protected]
```

Getters and Setters.

A Poisson random variable

3.3.2.3 x_chi

```
template<typename Generator >
std::chi_squared_distribution<double> cir_glasserman< Generator >::x_chi  [protected]
```

A Chi squared random variable

3.3.2.4 z_norm

```
template<typename Generator >
std::normal_distribution<double> cir_glasserman< Generator >::z_norm  [protected]
```

A standard Normal random variable

The documentation for this class was generated from the following file:

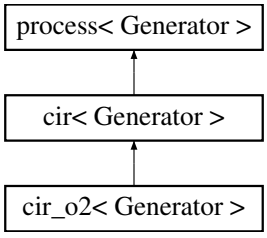
- cir.h

3.4 cir_o2< Generator > Class Template Reference

A scheme of order 2 for the CIR process.

```
#include <cir.h>
```

Inheritance diagram for cir_o2< Generator >:



Public Member Functions

- cir_o2 ()
  Constructor.
- cir_o2 (double state_0, double k, double a, double s)
  Constructor.
- ~cir_o2 ()
  Destructor.
- std::vector< double > next_step (Generator &gen, std::vector< double > state, double t)
  Pure virtual function. Will be implemented in the different schemes.

Protected Attributes

- std::uniform_real_distribution< double > u
- normal_five_moments< Generator > y

3.4.1 Detailed Description

template<typename Generator>
class cir_o2< Generator >

A scheme of order 2 for the CIR process.

This class implements the Ninomiya-Victoir scheme for the CIR. It is of order 2.

3.4.2 Member Data Documentation

3.4.2.1 u

```
template<typename Generator >
std::uniform_real_distribution<double> cir_o2< Generator >::u  [protected]
```

A uniform variable on [0,1]

3.4.2.2 y

```
template<typename Generator >
normal_five_moments<Generator> cir_o2< Generator >::y  [protected]
```

A random variable with the same first five moments as a standard Gaussian

The documentation for this class was generated from the following file:

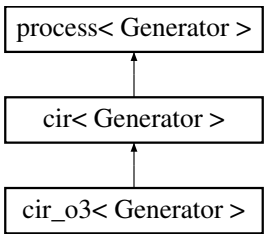
- cir.h

3.5 cir_o3< Generator > Class Template Reference

A scheme of order 3 for the CIR process.

`#include <cir.h>`

Inheritance diagram for cir_o3< Generator >:



Public Member Functions

- cir_o3 ()
  Constructor.
- cir_o3 (double state_0, double k, double a, double s)
  Constructor.
- ~cir_o3 ()
  Destructor.
- std::vector< double > next_step (Generator &gen, std::vector< double > state, double t)
  Pure virtual function. Will be implemented in the different schemes.

Protected Attributes

- std::uniform_real_distribution< double > u
- normal_seven_moments< Generator > seven
- zeta< Generator > zet
- rademacher< Generator > rad

3.5.1 Detailed Description

template<typename Generator>
class cir_o3< Generator >

A scheme of order 3 for the CIR process.

This class implements the Ninomiya-Victoir scheme for the CIR. It is of order 3.

3.5.2 Member Data Documentation

3.5.2.1 rad

```
template<typename Generator >
rademacher<Generator> cir_o3< Generator >::rad  [protected]
```

A Rademacher random variable

3.5.2.2 seven

```
template<typename Generator >
normal_seven_moments<Generator> cir_o3< Generator >::seven  [protected]
```

A random variable with the same first seven moments as a standard Gaussian

3.5.2.3 u

```
template<typename Generator >
std::uniform_real_distribution<double> cir_o3< Generator >::u  [protected]
```

A uniform variable on [0,1]

3.5.2.4 zet

```
template<typename Generator >
zeta<Generator> cir_o3< Generator >::zet  [protected]
```

A uniform variable on {1,2,3}

The documentation for this class was generated from the following file:

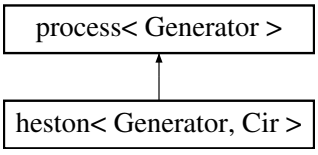
- cir.h

3.6 heston< Generator, Cir > Class Template Reference

Generation of a class with a Heston process.

```
#include <heston.h>
```

Inheritance diagram for heston< Generator, Cir >:



Public Member Functions

- heston ()
  Constructor.
- ~heston ()
  Destructor.
- heston (double cir_0, double x_0, double a, double k, double sigma, double rho, double r)
  Constructor.
- std::vector< double > **hw** (Generator &gen, std::vector< double > state, double t)
- std::vector< double > **hz** (Generator &gen, std::vector< double > state, double t)
- std::vector< double > next_step (Generator &gen, std::vector< double > state, double t)
  Function Next Step.
- double log_spot_one (double t)

Additional Inherited Members

3.6.1 Detailed Description

template<typename Generator, typename Cir>
class heston< Generator, Cir >

Generation of a class with a Heston process.

The class heston implements the class process. We compute the classical Heston model, along with the integrals of the price and the vol.

3.6.2 Member Function Documentation

3.6.2.1 log_spot_one()

```
template<typename Generator , typename Cir >
double heston< Generator, Cir >::log_spot_one (
            double t )
```

Computes the exact mean of log(S_t) where S is the price process. Useful for variance reduction by a control variate technique.

3.6.2.2 next_step()

```
template<typename Generator , typename Cir >
std::vector< double > heston< Generator, Cir >::next_step (
            Generator & gen,
            std::vector< double > state,
            double t )  [virtual]
```

Function Next Step.

Pure virtual function that will be over-written for every instance of process. Computes the value of the discretized process at time t + time_step. The value of the discretized process at t is stored in state.

Implements process< Generator >.

The documentation for this class was generated from the following file:

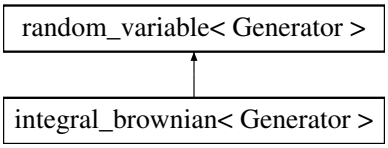
- heston.h

3.7 integral_brownian< Generator > Class Template Reference

A class to simulate the integral of a Brownian Motion with a classical Euler scheme.

`#include <integral_brownian.h>`

Inheritance diagram for integral_brownian< Generator >:



Public Member Functions

- **integral_brownian** (unsigned int number_steps)
- double operator() (Generator &gen)

  Virtual function.

Additional Inherited Members

3.7.1 Detailed Description

template<typename Generator>
class integral_brownian< Generator >

A class to simulate the integral of a Brownian Motion with a classical Euler scheme.

Mainly used for testing.

The documentation for this class was generated from the following file:

- integral_brownian.h

3.8 monte_carlo< Generator > Class Template Reference

A general class for Monte-Carlo simulation.

`#include <monte_carlo.h>`

Public Member Functions

- monte_carlo ()
  Constructors & destructors.
- monte_carlo (random_variable< Generator > *rv)
- double get_precision ()
  Getter.
- unsigned int get_cap ()
  Getter.
- double get_mean ()
  Getter.
- double get_std ()
  Getter.
- std::pair< double, double > get_confidence_interval ()
  Getter.
- void set_precision (double p)
  Setter.
- void set_cap (unsigned int c)
  Setter.
- double simulate (Generator &gen)
  Simulation of one realisation of the random variable.
- void compute (Generator &gen)
  Computations of a naive Monte Carlo estimator.
- void print ()
  Printing the results.

Public Attributes

- random_variable< Generator > * **variable**
- bool **computed**
- bool **successful**
- double **precision**
- unsigned int **cap_iterations**
- unsigned int **count**
- double **empirical_mean**
- double **empirical_var**
- double **empirical_std**
- double **ci_l_bound**
- double **ci_h_bound**

3.8.1 Detailed Description

template<typename Generator>
class monte_carlo< Generator >

A general class for Monte-Carlo simulation.

In this class we compute naive MC estimators (with confidence interval) given a random variable. We also implement different variance reduction techniques.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 monte_carlo() [1/2]

```
template<typename Generator >
monte_carlo< Generator >::monte_carlo ( )
```

Constructors & destructors.

All this are default values

By default, the variable we integrate is a N(0, 1)

3.8.2.2 monte_carlo() [2/2]

```
template<typename Generator >
monte_carlo< Generator >::monte_carlo (
            random_variable< Generator > * rv )
```

The random variable we want to integrate

All these are default values.

3.8.3 Member Function Documentation

3.8.3.1 compute()

```
template<typename Generator >
void monte_carlo< Generator >::compute (
            Generator & gen )
```

Computations of a naive Monte Carlo estimator.

This is the method that computes a naive MC estimator

3.8.3.2 print()

```
template<typename Generator >
void monte_carlo< Generator >::print ( )
```

Printing the results.

Printing our results properly

The documentation for this class was generated from the following file:

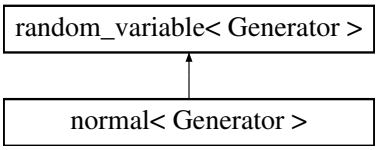
- monte_carlo.h

3.9 normal< Generator > Class Template Reference

A class to simulate a normal random variable.

`#include <normal.h>`

Inheritance diagram for normal< Generator >:



Public Member Functions

- **normal** (double m, double s)
- double operator() (Generator &gen)

  Virtual function.

Additional Inherited Members

3.9.1 Detailed Description

template<typename Generator>
class normal< Generator >

A class to simulate a normal random variable.

The documentation for this class was generated from the following file:

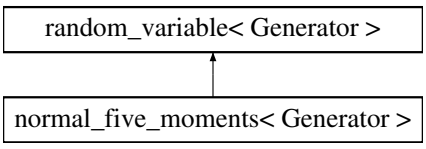
- normal.h

3.10 normal_five_moments< Generator > Class Template Reference

A class to simulate a random variable with the same first five moments as a standard Gaussian.

`#include <normal.h>`

Inheritance diagram for normal_five_moments< Generator >:



Public Member Functions

- double operator() (Generator &gen)
  Virtual function.

Additional Inherited Members

3.10.1 Detailed Description

template<typename Generator>
class normal_five_moments< Generator >

A class to simulate a random variable with the same first five moments as a standard Gaussian.

The documentation for this class was generated from the following file:

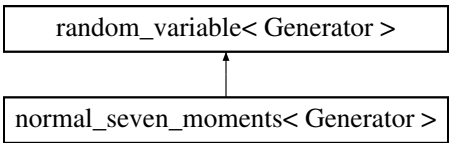
- normal.h

3.11 normal_seven_moments< Generator > Class Template Reference

A class to simulate a random variable with the same first seven moments as a standard Gaussian.

`#include <normal.h>`

Inheritance diagram for normal_seven_moments< Generator >:



Public Member Functions

- double operator() (Generator &gen)
  Virtual function.

Additional Inherited Members

3.11.1 Detailed Description

template<typename Generator>
class normal_seven_moments< Generator >

A class to simulate a random variable with the same first seven moments as a standard Gaussian.

The documentation for this class was generated from the following file:

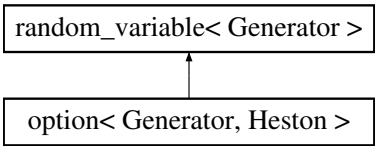
- normal.h

3.12 option< Generator, Heston > Class Template Reference

A class to code an option.

`#include <option.h>`

Inheritance diagram for option< Generator, Heston >:



Public Member Functions

- option ()
  Constructor.
- ~option ()
  Destructor.
- option (double maturity, double strike, double cir_0, double x_0, double a, double k, double sigma, double rho, double r, char type)
  Constructor.
- double payoff (std::vector< std::vector< double >> trajectory)
  Computes the payoff of the option.
- double test_log_spot_one (std::vector< std::vector< double >> trajectory)
  Function to test log_spot_one.
- double operator() (Generator &gen)
  Virtual function.
- void set_num_steps (unsigned int n)
  Setter.
- double theoretical_log_spot_one ()
  Getter.

Additional Inherited Members

3.12.1 Detailed Description

template<typename Generator, typename Heston>
class option< Generator, Heston >

A class to code an option.

The option can be both European or Asian.

The documentation for this class was generated from the following file:

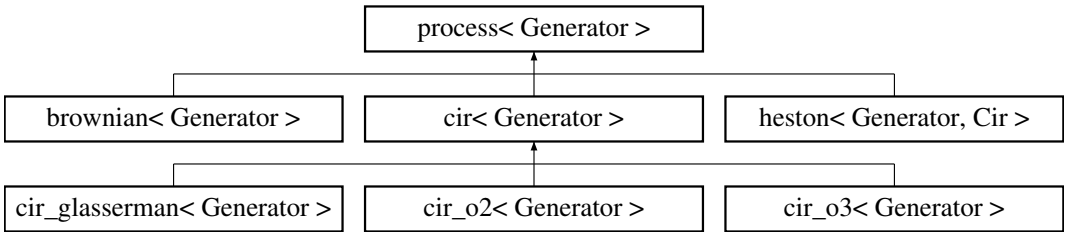
- option.h

3.13 process< Generator > Class Template Reference

A general class to code the discretization of a stochastic process on the time interval [0,T].

`#include <process.h>`

Inheritance diagram for process< Generator >:



Public Member Functions

- process ()
  Constructor.
- ~process ()
  Destructor.
- void set_time_end (double t)
  Setter.
- void set_num_steps (unsigned int n)
  Setter.
- void set_state_0 (std::vector< double > s)
  Setter.
- void set_time_grid ()
  Setter.
- void set_state (std::vector< double > s)
  Setter.
- unsigned int get_num_steps ()
  Getter.
- std::vector< double > get_state_0 ()
  Getter.
- double get_time_end ()
  Getter.
- double get_time_step ()
  Getter.
- void write_to_plot (Generator &gen, unsigned int num_plots, int coordinate)
  For plotting.
- virtual std::vector< double > next_step (Generator &gen, std::vector< double > state, double t)=0
  Function Next Step.
- std::vector< std::vector< double > > operator() (Generator &gen)
  Operator()
- void print_trajectory (int coordinate)
  Prints the results of the discretization.

Protected Attributes

- unsigned int **dimension**
- double time_step
- double time_end
- unsigned int num_steps
- std::vector< double > state_0
- std::vector< double > **state**
- std::vector< double > **time_grid**
- std::vector< std::vector< double > > trajectory
- bool **computed**

3.13.1 Detailed Description

template<typename Generator>
class process< Generator >

A general class to code the discretization of a stochastic process on the time interval [0,T].

The discretization scheme can be fixed either by chosing the time horizon (time_end) or the number of steps (num_steps).

3.13.2 Member Function Documentation

3.13.2.1 next_step()

```
template<typename Generator >
virtual std::vector<double> process< Generator >::next_step (
            Generator & gen,
            std::vector< double > state,
            double t )  [pure virtual]
```

Function Next Step.

Pure virtual function that will be over-written for every instance of process. Computes the value of the discretized process at time t + time_step. The value of the discretized process at t is stored in state.

Implemented in cir_glasserman< Generator >, cir_o3< Generator >, cir_o2< Generator >, heston< Generator, Cir >, cir< Generator >, and brownian< Generator >.

3.13.2.2 operator()()

```
template<typename Generator >
std::vector< std::vector< double > > process< Generator >::operator() (
            Generator & gen )
```

Operator()

Overloads the operator (). Allows to draw a new instance of the discretized process.

3.13.2.3 write_to_plot()

```
template<typename Generator >
void process< Generator >::write_to_plot (
            Generator & gen,
            unsigned int num_plots,
            int coordinate )
```

For plotting.

Computes and plots num_plot instances of the discretized process. The method writes the relevant data in a .dat file and calls gnuplot for plotting

3.13.3 Member Data Documentation

3.13.3.1 num_steps

```
template<typename Generator >
unsigned int process< Generator >::num_steps  [protected]
```

Number of steps in time discretization

3.13.3.2 state_0

```
template<typename Generator >
std::vector<double> process< Generator >::state_0  [protected]
```

Initial value for the process at time t = 0

3.13.3.3 time_end

```
template<typename Generator >
double process< Generator >::time_end  [protected]
```

Simulation between 0 and time_end

3.13.3.4 time_step

```
template<typename Generator >
double process< Generator >::time_step  [protected]
```

Time step of the discretization

3.13.3.5 trajectory

```
template<typename Generator >
std::vector<std::vector<double> > process< Generator >::trajectory  [protected]
```

Vector of vectors used to compute the values taken by the process This choice is motivated by the mutli-dimensional Heston model

The documentation for this class was generated from the following file:

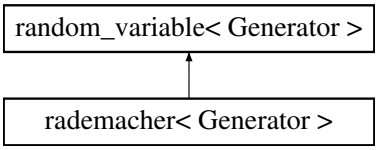
- process.h

3.14 rademacher< Generator > Class Template Reference

A Rademacher random variable.

```
#include <normal.h>
```

Inheritance diagram for rademacher< Generator >:



Public Member Functions

- double operator() (Generator &gen)

  Virtual function.

Additional Inherited Members

3.14.1 Detailed Description

template<typename Generator>
class rademacher< Generator >

A Rademacher random variable.

The documentation for this class was generated from the following file:

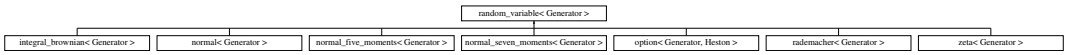
- normal.h

3.15 random_variable< Generator > Class Template Reference

A general class to implement a random variable.

`#include <random_variable.h>`

Inheritance diagram for random_variable< Generator >:



Public Member Functions

- random_variable ()
  Constructor.
- ~random_variable ()
  Destructor.
- void print ()
  Printing.
- virtual double operator() (Generator &gen)=0
  Virtual function.

Protected Attributes

- bool **realised**
- double **realisation**

3.15.1 Detailed Description

template<typename Generator>
class random_variable< Generator >

A general class to implement a random variable.

The documentation for this class was generated from the following file:

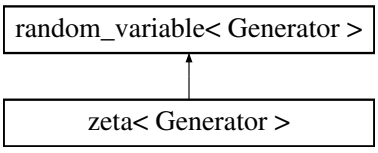
- random_variable.h

3.16 zeta< Generator > Class Template Reference

A uniform random variable on {1,2,3}.

`#include <normal.h>`

Inheritance diagram for zeta< Generator >:



Public Member Functions

- double operator() (Generator &gen)
  Virtual function.

Additional Inherited Members

3.16.1 Detailed Description

template<typename Generator>
class zeta< Generator >

A uniform random variable on {1,2,3}.

The documentation for this class was generated from the following file:

- normal.h

Index