proba\_num

Generated by Doxygen 1.8.14

# **Contents**

1	Hier	archica	I Index	1	
	1.1	Class I	Hierarchy	1	
2	Clas	s Index		3	
	2.1	Class I	List	3	
3	Clas	s Docu	S Documentation S		
	3.1	browni	an < Generator > Class Template Reference	5	
		3.1.1	Detailed Description	5	
		3.1.2	Constructor & Destructor Documentation	6	
			3.1.2.1 brownian()	6	
	3.2	cir< G	enerator > Class Template Reference	6	
		3.2.1	Detailed Description	7	
		3.2.2	Member Data Documentation	7	
			3.2.2.1 a	7	
			3.2.2.2 k	7	
			3.2.2.3 sigma	7	
	3.3	cir_glasserman< Generator > Class Template Reference		8	
		3.3.1	Detailed Description	8	
		3.3.2	Member Data Documentation	8	
			3.3.2.1 d	9	
			3.3.2.2 n_poisson	9	
			3.3.2.3 x_chi	9	
			3324 z norm	q	

ii CONTENTS

3.4	cir_o2	< Generator > Class Template Reference	9
	3.4.1	Detailed Description	10
	3.4.2	Member Data Documentation	10
		3.4.2.1 u	10
		3.4.2.2 y	10
3.5	cir_o3	< Generator > Class Template Reference	11
	3.5.1	Detailed Description	11
	3.5.2	Member Data Documentation	11
		3.5.2.1 rad	12
		3.5.2.2 seven	12
		3.5.2.3 u	12
		3.5.2.4 zet	12
3.6	heston	n < Generator, Cir > Class Template Reference	12
	3.6.1	Detailed Description	13
	3.6.2	Member Function Documentation	13
		3.6.2.1 log_spot_one()	13
		3.6.2.2 next_step()	13
3.7	integra	al_brownian< Generator > Class Template Reference	14
	3.7.1	Detailed Description	14
3.8	monte_	_carlo< Generator > Class Template Reference	14
	3.8.1	Detailed Description	15
	3.8.2	Constructor & Destructor Documentation	16
		3.8.2.1 monte_carlo() [1/2]	16
		3.8.2.2 monte_carlo() [2/2]	16
	3.8.3	Member Function Documentation	16
		3.8.3.1 compute()	16
		3.8.3.2 print()	16
3.9	norma	I< Generator > Class Template Reference	17
	3.9.1	Detailed Description	17
3.10	.10 normal_five_moments< Generator > Class Template Reference		

CONTENTS

3.	10.1 Detailed Description	18
3.11 no	ormal_seven_moments < Generator > Class Template Reference	18
3.	11.1 Detailed Description	18
3.12 op	otion < Generator, Heston > Class Template Reference	19
3.	12.1 Detailed Description	19
3.13 pr	rocess < Generator > Class Template Reference	20
3.	13.1 Detailed Description	21
3.	13.2 Member Function Documentation	21
	3.13.2.1 next_step()	21
	3.13.2.2 operator()()	21
	3.13.2.3 write_to_plot()	22
3.	13.3 Member Data Documentation	22
	3.13.3.1 num_steps	22
	3.13.3.2 state_0	22
	3.13.3.3 time_end	22
	3.13.3.4 time_step	22
	3.13.3.5 trajectory	23
3.14 ra	demacher< Generator > Class Template Reference	23
3.	14.1 Detailed Description	23
3.15 ra	andom_variable < Generator > Class Template Reference	24
3.	15.1 Detailed Description	24
3.16 ze	eta < Generator > Class Template Reference	24
3.	16.1 Detailed Description	25
Index		27

## **Chapter 1**

# **Hierarchical Index**

## 1.1 Class Hierarchy

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

$monte\_carlo < Generator > \dots $	4
process< Generator >	20
brownian < Generator >	5
$cir < Generator > \ \ldots \ \ldots \ \ldots \ \ldots \ \ldots \ \ldots \ \ldots$	6
cir_glasserman< Generator >	8
cir_o2< Generator >	9
cir_o3< Generator >	1
heston< Generator, Cir >	2
random_variable < Generator >	!4
integral_brownian < Generator >	4
normal < Generator >	7
normal_five_moments< Generator >	7
normal_seven_moments< Generator >	8
option < Generator, Heston >	9
rademacher < Generator >	23
zeta < Generator > 2	4

2 Hierarchical Index

## Chapter 2

# **Class Index**

## 2.1 Class List

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

brownian< Generator >	
A class to discretize a Brownian Motion with a classical Euler scheme	5
cir< Generator >	
Generation of a class with a CIR process	6
cir_glasserman< Generator >	
An exact scheme for the CIR process	8
cir_o2< Generator >	
A scheme of order 2 for the CIR process	9
cir_o3< Generator >	
A scheme of order 3 for the CIR process	11
heston< Generator, Cir >	
Generation of a class with a Heston process	12
integral_brownian < Generator >	
A class to simulate the integral of a Brownian Motion with a classical Euler scheme	14
monte_carlo< Generator >	
A general class for Monte-Carlo simulation	14
normal < Generator >	4-
A class to simulate a normal random variable	17
normal_five_moments < Generator >	
A class to simulate a random variable with the same first five moments as a standard Gaussian	17
normal_seven_moments < Generator >	40
A class to simulate a random variable with the same first seven moments as a standard Gaussian	16
option< Generator, Heston >  A class to code an option	19
	18
process < Generator >  A general class to code the discretization of a stochastic process on the time interval [0,T]	20
rademacher < Generator >	20
A Rademacher random variable	23
random variable < Generator >	20
A general class to implement a random variable	24
zeta < Generator >	
A uniform random variable on {1.2.3}	24

4 Class Index

## **Chapter 3**

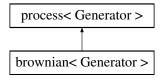
## **Class Documentation**

## 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

 $\label{template} \begin{tabular}{ll} template < type name Generator > \\ class brownian < Generator > \\ \end{tabular}$ 

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:

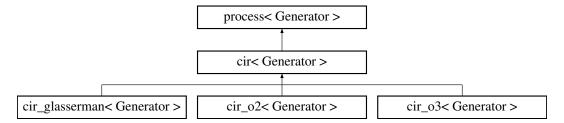
· brownian.h

## 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:

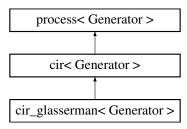
• cir.h

## 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
- $\bullet \ \, std::normal\_distribution < double > {\color{red} z\_norm}$
- 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:

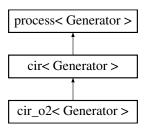
• cir.h

## 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:

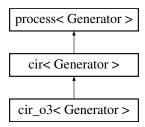
· cir.h

## 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:

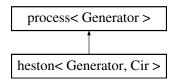
• cir.h

## 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()

## 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:

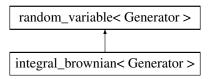
· heston.h

## 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)

Sattor

• 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

The random variable we want to integrate

All these are default values.

## 3.8.3 Member Function Documentation

## 3.8.3.1 compute()

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:

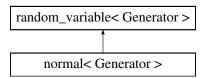
· monte\_carlo.h

## 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

 $\label{template} \begin{tabular}{ll} template < type name Generator > \\ class normal < Generator > \\ \end{tabular}$ 

A class to simulate a normal random variable.

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

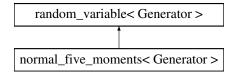
normal.h

## 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:

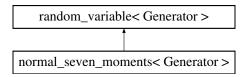
· normal.h

## 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:

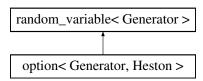
· normal.h

## 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:

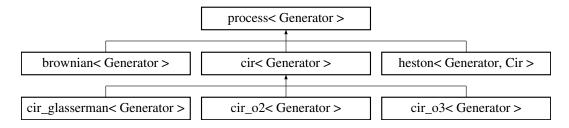
option.h

## 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 cess.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.

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()

## 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()()

#### Operator()

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

### 3.13.2.3 write\_to\_plot()

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:

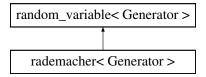
· process.h

## 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:

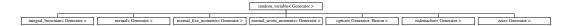
· normal.h

## 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:

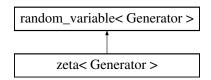
random\_variable.h

## 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

 $\label{template} \begin{tabular}{ll} template < typename Generator > \\ class zeta < Generator > \\ \end{tabular}$ 

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

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

• normal.h

# Index

a	monte_carlo< Generator >, 14
cir, 7	
	n_poisson
brownian	cir_glasserman, 9
brownian, 6	next_step
brownian < Generator >, 5	heston, 13
	process, 21
cir	normal $<$ Generator $>$ , 17
a, 7	normal_five_moments< Generator >, 17
k, 7	normal_seven_moments< Generator >, 1
sigma, 7	num_steps
cir< Generator >, 6	process, 22
cir_glasserman	p. 66666, 22
d, 8	operator()
n_poisson, 9	process, 21
x_chi, 9	option< Generator, Heston >, 19
	option Concrator, restor >, 10
z_norm, 9	print
cir_glasserman< Generator >, 8	monte carlo, 16
cir_o2	<del>-</del> · · ·
u, 10	process
y, 10	next_step, 21
cir_o2< Generator >, 9	num_steps, 22
cir_o3	operator(), 21
rad, 11	state_0, 22
seven, 12	time_end, 22
u, 12	time_step, 22
zet, 12	trajectory, <mark>22</mark>
cir_o3< Generator >, 11	write_to_plot, 21
compute	process< Generator >, 20
monte_carlo, 16	
	rad
d	cir_o3, 11
cir_glasserman, 8	rademacher < Generator >, 23
on_glassorman, o	random_variable< Generator >, 24
heston	_ ,
log_spot_one, 13	seven
next_step, 13	cir o3, 12
	sigma
heston< Generator, Cir >, 12	cir, 7
integral_brownian< Generator >, 14	state 0
integral_brownlan< denerator >, 14	process, 22
k	process, 22
	time end
cir, 7	process, 22
log enet one	•
log_spot_one	time_step
heston, 13	process, 22
monto carlo	trajectory
monte_carlo	process, 22
compute, 16	
monte_carlo, 16	u 
print, 16	cir o2. 10

28 INDEX

```
cir_o3, 12

write_to_plot
    process, 21

x_chi
    cir_glasserman, 9

y
    cir_o2, 10

z_norm
    cir_glasserman, 9

zet
    cir_o3, 12

zeta < Generator >, 24
```