Option pricing and analysis

Generated by Doxygen 1.8.4

Fri Dec 12 2014 00:37:00

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

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

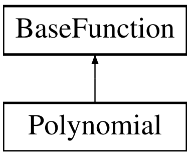
Class Documentation

4.1 BaseFunction Class Reference

common interface for base functions to fit 1d function

```
#include <base_function.h>
```

Inheritance diagram for BaseFunction:



Public Member Functions

- BaseFunction (int n)

  initialize base function with a certain order n
- virtual double operator() (double x) const =0

  evaulate value at x
- virtual BaseFunction ∗ **clone** () const =0

Protected Attributes

- int **order_**

4.1.1 Detailed Description

common interface for base functions to fit 1d function

The base functions can be polynomials, Fourier series, etc. They can be ordered by an integer n.

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

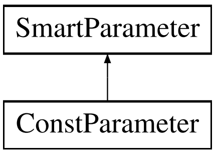
- src/lib/base_function.h

4.2 ConstParameter Class Reference

constant parameter class

`#include <market_parameters.h>`

Inheritance diagram for ConstParameter:



Public Member Functions

- **ConstParameter** (double value)
- double operator() (double t=0) const

  get value at a give time
- double Integral (double time1, double time2) const

  evaluate integral at interval (time1, time2)
- double IntegralSquare (double time1, double time2) const

  evaluate integral of square
- SmartParameter ∗ clone () const

  virtual constructor

4.2.1 Detailed Description

constant parameter class

the market parameters are constant with respect to time

4.2.2 Member Function Documentation

4.2.2.1 double ConstParameter::Integral (double *time1,* double *time2*) const `[inline], [virtual]`

evaluate integral at interval (time1, time2)

Parameters

time1	left range of time interval
time2	right range of time interval

Implements SmartParameter.

4.2.2.2 double ConstParameter::IntegralSquare (double *time1,* double *time2*) const `[inline], [virtual]`

evaluate integral of square

Int(x^2)

Implements SmartParameter.

4.2.2.3 double ConstParameter::operator() (double *t* = `0`) const `[inline], [virtual]`

get value at a give time

Parameters

t	time, default value is zero

Implements SmartParameter.

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

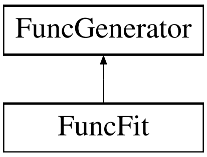
- src/lib/market_parameters.h

4.3 FuncFit Class Reference

give multiple observations (x0, y0), (x1, y1), ..., use bases function to fit the functional relationship between x and y.

`#include <func_generation.h>`

Inheritance diagram for FuncFit:



Public Member Functions

- bool AssimilateObs (const std::vector< double > &x, const std::vector< double > &y)

  Take in observation and do least square fit functional relationship between x and y.

Additional Inherited Members

4.3.1 Detailed Description

give multiple observations (x0, y0), (x1, y1), ..., use bases function to fit the functional relationship between x and y.

Does least square fit.

4.3.2 Member Function Documentation

4.3.2.1 bool FuncFit::AssimilateObs (const std::vector< double > & *x*, const std::vector< double > & *y*)

Take in observation and do least square fit functional relationship between x and y.

Parameters

x	observation for x
y	observation for y

Returns

bool whether assimilation is successful

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

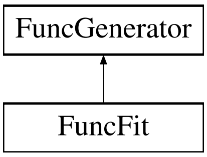
- src/lib/func_generation.h
- src/lib/func_generation.cc

4.4 FuncGenerator Class Reference

generate a function object (callable) from base functions with specified linear combination

```
#include <func_generation.h>
```

Inheritance diagram for FuncGenerator:



Public Member Functions

- FuncGenerator ()

  construct a function with a certain num of bases functions
- void set_coeffs (const std::vector< double > &coeffs)

  *set coeffs_*
- void add_base (BaseFunction ∗basef_ptr)

  set the ith base function
- double operator() (double x) const

  evaulate at x

Protected Attributes

- std::vector< std::unique_ptr
  < BaseFunction > > **base_funcs_**
- std::vector< double > coeffs_
- int **num_funcs_**
- bool is_initialized_

  whether ready to evaulate

4.4.1 Detailed Description

generate a function object (callable) from base functions with specified linear combination

4.4.2 Constructor & Destructor Documentation

4.4.2.1 FuncGenerator::FuncGenerator ( ) `[inline]`

construct a function with a certain num of bases functions

Parameters

num_bases	number of bases functionsa vector of base functions

4.4.3 Member Data Documentation

4.4.3.1 std::vector<double> FuncGenerator::coeffs_ `[protected]`

coefficients for linear combination, e.g., < A0 + A1∗x + A2∗x^2 + ...

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

- src/lib/func_generation.h
- src/lib/func_generation.cc

4.5 LeastSquareMC Class Reference

Public Member Functions

- **LeastSquareMC** (const PathGenerator &path_gen, const VanillaOption &van_option)
- double **DoSimulation** (double spot, int num_paths=100)

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

- src/lib/least_square_mc.h
- src/lib/least_square_mc.cc

4.6 MarketParameters Class Reference

a class that combines market parameters together and thus provides a uniform interface.

```
#include <market_parameters.h>
```

Public Member Functions

- **MarketParameters** (const SmartParameter &sigma, const SmartParameter &r, const SmartParameter &d)
- **MarketParameters** (const MarketParameters &orig_params)
- MarketParameters & **operator=** (const MarketParameters &orig_params)

Public Attributes

- SmartParameter ∗ **volatility_**
- SmartParameter ∗ **interest_rate_**
- SmartParameter ∗ **divident_rate_**

4.6.1 Detailed Description

a class that combines market parameters together and thus provides a uniform interface.

In current implemetation, this class contains just volatility, interest rate and divident rate.

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

- src/lib/market_parameters.h
- src/lib/market_parameters.cc

4.7 Matrix2d Class Reference

A two dimensional matrix class indexed as M(i,j) and provides many element-wise array operations: +, -, ∗, /, and matrix operations: dot, left_divide.

```
#include <matrix2d.h>
```

Public Member Functions

- Matrix2d (int num_rows, int num_colmns, double init_val=0.0)

  allocate space for a matrix with specified size and initialization value
- Matrix2d (const std::vector< std::vector< double > > &array_in)

  initialize a matrix using vector<vector<double> >
- Matrix2d (std::vector< std::vector< double > > &&array_in)

  move semantics: initialize with rvalue vector<vector<double> >
- Matrix2d (const std::vector< double > &array_in)

  create a Nx1 matrix, which is often appears on the rhs of Ax = b
- Matrix2d (const Matrix2d &matrix_in)

  copy constructor
- Matrix2d (Matrix2d &&matrix_in)

  move semantics: initialize with rvalue Matrix2d object
- Matrix2d & operator= (const Matrix2d &matrix_in)

  copy assgiment operator
- Matrix2d & operator= (Matrix2d &&matrix_in)

  move semantics: assgin with rvalue Matrix2d object
- Matrix2d & operator= (double value)

  asign all elements in the matrix to a single value
- double & operator() (int i_row, int j_colmn)

  access or assign an element in a matrix as A(i,j)
- double operator() (int i_row, int j_colmn) const

  access a const matrix
- unsigned int get_num_rows () const

  returns the number of rows
- unsigned int get_num_columns () const

  return the number of column
- Matrix2d operator+ (const Matrix2d &matrix_in)

  add two matrixes
- Matrix2d & **operator+=** (const Matrix2d &matrix_in)
- Matrix2d & **operator+=** (double scalar)
- Matrix2d operator- (const Matrix2d &matrix_in)

  substract two matrixes
- Matrix2d & **operator-=** (const Matrix2d &matrix_in)
- Matrix2d & **operator-=** (double scalar)
- Matrix2d operator- ()
- Matrix2d operator* (const Matrix2d &matrix_in)

  multiple each pair of elment in the same position in two matrixes
- Matrix2d & **operator*=** (const Matrix2d &matrix_in)
- Matrix2d & operator*= (double scalar)
- Matrix2d **operator/** (const Matrix2d &matrix_in)
- Matrix2d & **operator/=** (const Matrix2d &matrix_in)
- Matrix2d & **operator/=** (double scalar)
- Matrix2d dot (const Matrix2d &matrix_in)

  matrix multiplication
- Matrix2d **transpose** ()
- Matrix2d left_divide (const Matrix2d &b)

  solve linear equation; similar to \ in Matlab
- double det ()

  return determinant of the matrix

Friends

- Matrix2d operator+ (const Matrix2d &rhs, double scalar)
- Matrix2d **operator+** (double scalar, const Matrix2d &rhs)
- Matrix2d operator- (const Matrix2d &rhs, double scalar)

  substract a scalar from a matrix
- Matrix2d **operator-** (double scalar, const Matrix2d &rhs)
- Matrix2d operator∗ (const Matrix2d &rhs, double scalar)

  multiply a matrix by a scalar
- Matrix2d **operator∗** (double scalar, const Matrix2d &rhs)
- Matrix2d **operator/** (const Matrix2d &rhs, double scalar)
- Matrix2d **operator/** (double scalar, const Matrix2d &rhs)
- std::ostream & operator<< (std::ostream &os, const Matrix2d &matrix1)

  overload insertion operator for easy display to screen
- bool check_if_same_size (const Matrix2d &matrix1, const Matrix2d &matrix2)

4.7.1 Detailed Description

A two dimensional matrix class indexed as M(i,j) and provides many element-wise array operations: +, -, ∗, /, and matrix operations: dot, left_divide.

Note

matrix indexing starts from 0
binary operations are returned by value. Should turn on optimization to use RVO.

Example usage

```
Initialize a matrix
  Matrix2d M(5, 5, 1.0); //5x5 matrix with every element to be 1.0
  Matrix2d D(3, 2, 0.0); //3x2 matrix with every element to be 0.0

Access and assign value to a particular element
  M(0,0) = 3.14;

Matrix operations with scalar
  Matrix2d N = M*2.0;
  Matrix2d C = 1.5 / M;
  N += 1.0;

Elementwise matrix operation
  D = M + C;
  D = M * C;

Matrix multiply
  D = M.dot(C);

Evaulate determinant
  double d = D.det();

Solve linear equation: A x = b
  b = Matrix2d (5, 1, 1.0);
  Matrix2d x = M.left_divide(b);
```

4.7.2 Constructor & Destructor Documentation

4.7.2.1 Matrix2d::Matrix2d (int *num_rows*, int *num_colmns*, double *init_val* = 0.0)

allocate space for a matrix with specified size and initialization value

Parameters

num_rows	number of rows
num_colmns	number of columns
init_val	value at initilization, default is 0.0

4.7.3 Member Function Documentation

4.7.3.1 double Matrix2d::det ( )

return determinant of the matrix

determinant is evaulated after LU decomposition; and whether it has been evaulated is indicated by LU_if_initialized-_

Note

> only works for square matrix; throw an exception is performed for non-square matrix

4.7.3.2 Matrix2d Matrix2d::dot (const Matrix2d & *matrix_in*)

matrix multiplication

lhs(i,k) = sum(rhs1(i,j)∗(j,k)) over j

4.7.3.3 Matrix2d Matrix2d::left_divide (const Matrix2d & *b*)

solve linear equation; similar to \ in Matlab

solves equation A x = b

for Ax = b, x = A.left_divide(b)

Parameters

rhs	b on the rhs of the equation. Number of rows must equal that of A Can have multiple columns Algorithm: A is first decomposed into A = L U then L y = b is solved, and finally U x = y is solved

4.7.3.4 Matrix2d Matrix2d::operator∗ (const Matrix2d & *matrix_in*)

multiple each pair of elment in the same position in two matrixes

lhs(i,j) = rhs1(i,j)∗rhs2(i,j)

Note

> not real matrix multiplication, which is dot operation below

4.7.3.5 Matrix2d & Matrix2d::operator∗= (double *scalar*)

multiply U in LU decomp. by scalar

4.7.3.6 Matrix2d Matrix2d::operator- ( )

unary operation: returns a matrix whose every element is negative of that in original matrix change the sign of U in LU decomp; no need to cal LU again

4.7.4 Friends And Related Function Documentation

4.7.4.1 bool check_if_same_size (const Matrix2d & *matrix1*, const Matrix2d & *matrix2*) `[friend]`

check if two matrices have the same dimension returns true if of same dimension

4.7.4.2 Matrix2d operator+ (const Matrix2d & *rhs*, double *scalar*) `[friend]`

add a matrix with a scalar, which mean add the scalar to each element in the matrix

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

- src/lib/matrix2d.h
- src/lib/matrix2d.cc

4.8 ParkMillerOneRand Class Reference

Generate one random integer using Park Miller congruential generator Need a none zero seed; default seed is 1.

`#include <park_miller_rand.h>`

Public Member Functions

- **ParkMillerOneRand** (long seed=1)
- long GetOneRandInt ()

  returns one random integer
- void **set_seed** (long seed)
- unsigned long get_range_max ()

  return the upper bound for random number
- unsigned long get_range_min ()

  return the lower bound for random number

4.8.1 Detailed Description

Generate one random integer using Park Miller congruential generator Need a none zero seed; default seed is 1.

long integer on the platform must at least be 32 bit

4.8.2 Member Function Documentation

4.8.2.1 unsigned long ParkMillerOneRand::get_range_max ( )

return the upper bound for random number

The random number generated is within a min and maxmum bound. This is [1, 2147483646] in this implementation.

4.8.2.2 long ParkMillerOneRand::GetOneRandInt ( )

returns one random integer

Lehmer random number generator (RNG) or Park-Miler RNG. It is a congruential generator.

Park-Miller Algorithm:

$$X_{k+1} = A \cdot X_k \bmod M$$

Based on Diane Crawford (1993, Technical correspondence, Communications of the ACM, Vol 36), the parameters are choosen as A = 48271 and M = 2147483647 (which is 2^31 - 1).

Schrage's Algorithm: To aviod multiplication of 32-bit numbers on a 32-bit machine, Schrage's algo is used. It is based on an approximate factorization of M as

$$M = AQ + R,\ Q = [M/A],\ R = M \bmod A$$

The brackets [] denotes integer division. We want R to be small and choose Q = 44488 and R = 3399. Note R $<$ A and R $<$ Q. Then the iteration becomes

$$\begin{aligned} X_{k+1} &= A(I_i - [X_i/q] \cdot Q) - R \cdot [X_i/Q] \\ &= A(I_i \bmod Q) - R \cdot [X_i/Q] \end{aligned}$$

If $I_{k+1} < 0$, then

$$X_{k+1} = X_{k+1} + m$$

Proof: the key to prove is that $x \bmod b = x - [x/b]b$. As M = AQ + R, we have

$$\begin{aligned} X_{k+1} &= AX_k - [\tfrac{AX_k}{AQ+R}](AQ+R) \\ &= AX_k - [\tfrac{X_k}{Q}\tfrac{1}{1+R/(AQ)}](AQ+R) \end{aligned}$$

Because R/AQ $<<$ 1, use Taylor expansion, we have

$$X_{k+1} = AX_k - [\frac{X_k}{Q} - \frac{X_k}{AQ}\frac{R}{Q}](AQ+R)$$

Because AQ∼M, R<Q, we have $\frac{X_k}{AQ}\frac{R}{Q} < 1$. So $[\frac{X_k}{Q} - \frac{X_k}{AQ}\frac{R}{Q}]$ is either $[\frac{X_k}{Q}]$ or $[\frac{X_k}{Q}] - 1$

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

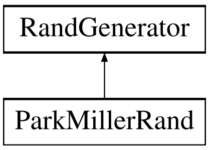
- src/lib/park_miller_rand.h
- src/lib/park_miller_rand.cc

4.9 ParkMillerRand Class Reference

Generate one or an array of uniformly/normally distributed random numbers.

```
#include <park_miller_rand.h>
```

Inheritance diagram for ParkMillerRand:



Public Member Functions

- ParkMillerRand (long seed=1)

  constructs a random number generator with a given seed (default is 1)
- RandGenerator ∗ clone () const

  bridge pattern
- void GenUniformRand (std::vector< double > &rand_array)

generate a uniformly distributed random number array within (0,1)

- void GenUniformRand (double &rand_num)

  generator one uniformly distributed random number within (0,1)
- void SkipNumOfPath (int num_of_path)
- void set_seed (int seed)

  set the seed for random number generator
- void Reset ()

  reset the generator to its initial state (when constructed)

4.9.1 Detailed Description

Generate one or an array of uniformly/normally distributed random numbers.

Note

> An adapter pattern to make class ParkMillerOneRand have the same interface as the base class RandGenerator

Usage: refer to base class RandGenerator

4.9.2 Constructor & Destructor Documentation

4.9.2.1 ParkMillerRand::ParkMillerRand (long *seed* = `1`)

constructs a random number generator with a given seed (default is 1)

Parameters

seed	default is 1

< + 1 thus cannot reach 1.0

4.9.3 Member Function Documentation

4.9.3.1 void ParkMillerRand::GenUniformRand (std::vector< double > & *rand_array*) `[virtual]`

generate a uniformly distributed random number array within (0,1)

Parameters

rand_array	a vector of double passed in by reference to store the output.

Note

> 0 and 1 are excluded.

Implements RandGenerator.

4.9.3.2 void ParkMillerRand::GenUniformRand (double & *rand_num*) `[virtual]`

generator one uniformly distributed random number within (0,1)

Parameters

rand_num	a double passed in by reference to store the output.

Implements RandGenerator.

4.9.3.3 void ParkMillerRand::SkipNumOfPath (int *num_of_path*) [virtual]

skip a certain number of random numbers in order to avoid same numbers (paths)

Implements RandGenerator.

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

- src/lib/park_miller_rand.h
- src/lib/park_miller_rand.cc

4.10 PathGenerator Class Reference

generator geometric brownian motion with parameters specified by MarketParameters object, and at specified times

`#include <path_generation.h>`

Public Member Functions

- PathGenerator (const MarketParameters &market_params, RandGenerator &rand_gen, double spot, int num_times, double expiration_time)

  generate geometric brownian motion at equally spaced times
- PathGenerator (const MarketParameters &market_parms, RandGenerator &rand_gen, double spot, const std::vector< double > &time_points)

  generate path at times specified by time_points
- **PathGenerator** (const PathGenerator &path_gen)
- PathGenerator & **operator=** (const PathGenerator &path_gen)
- void set_spot (double spot)

  set spot price (price at time 0)
- std::vector< double > **GetOnePath** ()
- std::vector< std::vector
  < double > > **GetNPaths** (int num_paths)
- std::vector< double > **get_time_points** ()

4.10.1 Detailed Description

generator geometric brownian motion with parameters specified by MarketParameters object, and at specified times

4.10.2 Constructor & Destructor Documentation

4.10.2.1 PathGenerator::PathGenerator (const **MarketParameters** & *market_params,* **RandGenerator** & *rand_gen,* double *spot,* int *num_times,* double *expiration_time*)

generate geometric brownian motion at equally spaced times

The browian motion is time series S0, S1, ... Sn. S0 is at time zero, and is equal to spot. n is num_times. Sn is at the expiration_time. The times are (1/N, 2/N, ..., 1) $*$ expiration_time, where N is num_times.

Parameters

params_in	contains volatility and interest rate
rand_gen	random number generator
spot	spot price at time 0
num_times	number of points for the generated path
expiration_time	how long does it expire from now (time 0)

4.10.2.2 PathGenerator::PathGenerator (const **MarketParameters** & *market_parms,* **RandGenerator** & *rand_gen,* double *spot,* const std::vector< double > & *time_points*)

generate path at times specified by time_points

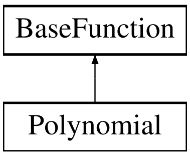
similar to the constructor above, but can generate path at unequally spaced times

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

- src/lib/path_generation.h
- src/lib/path_generation.cc

4.11 Polynomial Class Reference

Inheritance diagram for Polynomial:



Public Member Functions

- Polynomial (int n)

  initialize a polynomial with order n, that is x^n
- **Polynomial** (const Polynomial &poly)
- Polynomial & **operator=** (const Polynomial &poly)
- double operator() (double x) const

  evaulate value at x
- BaseFunction ∗ **clone** () const

Additional Inherited Members

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

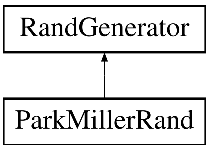
- src/lib/base_function.h

4.12 RandGenerator Class Reference

base class/interface for a suite of random generators

```
#include <rand_generator.h>
```

Inheritance diagram for RandGenerator:



Public Member Functions

- RandGenerator ()

    generate a RandGenerator which contains nothing
- virtual RandGenerator ∗ clone () const =0

    bridge pattern
- virtual void GenUniformRand (std::vector< double > &rand_array)=0

    generate a uniformly distributed random number array within (0,1)
- virtual void GenUniformRand (double &rand_num)=0

    generator one uniformly distributed random number within (0,1)
- virtual void SkipNumOfPath (int num_of_path)=0
- virtual void set_seed (int seed)=0

    set the seed for random number generator
- virtual void Reset ()=0

    reset the generator to its initial state (when constructed)
- virtual void GenNormRand (std::vector< double > &rand_array)

    generate an array of standard normal distribution
- virtual void GenNormRand (double &rand_num)

    generate one standard normal distribution

4.12.1 Detailed Description

base class/interface for a suite of random generators

Usage:

Overloaded for generating one random number and generating a vector of random numbers

4.12.2 Constructor & Destructor Documentation

4.12.2.1 RandGenerator::RandGenerator ( )

generate a RandGenerator which contains nothing

Note

> this version does not require a given dimensionality

4.12.3 Member Function Documentation

4.12.3.1 void RandGenerator::GenNormRand (std::vector< double > & *rand_array*) [virtual]

generate an array of standard normal distribution

Parameters

rand_array	a vector of double passed in by reference to store the output

if size of rand_array is an odd number, get a normal random number for rand_array[0] first, and then the size rest of the array is even

4.12.3.2 void RandGenerator::GenNormRand (double & *rand_num*) `[virtual]`

generate one standard normal distribution

Parameters

rand_num	a double passed in by reference to store the output.

4.12.3.3 virtual void RandGenerator::GenUniformRand (std::vector< double > & *rand_array*) `[pure virtual]`

generate a uniformly distributed random number array within (0,1)

Parameters

rand_array	a vector of double passed in by reference to store the output.

Note

0 and 1 are excluded.

Implemented in ParkMillerRand.

4.12.3.4 virtual void RandGenerator::GenUniformRand (double & *rand_num*) `[pure virtual]`

generator one uniformly distributed random number within (0,1)

Parameters

rand_num	a double passed in by reference to store the output.

Implemented in ParkMillerRand.

4.12.3.5 virtual void RandGenerator::SkipNumOfPath (int *num_of_path*) `[pure virtual]`

skip a certain number of random numbers in order to avoid same numbers (paths)

Implemented in ParkMillerRand.

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

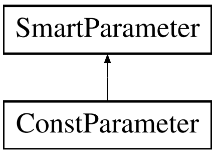
- src/lib/rand_generator.h
- src/lib/rand_generator.cc

4.13 SmartParameter Class Reference

class to manage market parameters including volatility and interest rate

`#include <market_parameters.h>`

Inheritance diagram for SmartParameter:



Public Member Functions

- virtual double operator() (double t=0) const =0

  get value at a give time
- virtual double Integral (double time1, double time2) const =0

  evaluate integral at interval (time1, time2)
- virtual double IntegralSquare (double time1, double time2) const =0

  evaluate integral of square
- virtual SmartParameter ∗ clone () const =0

  virtual constructor

4.13.1 Detailed Description

class to manage market parameters including volatility and interest rate

Volatility and interest rate are generally functions of time. Their integral and square integral are generally needed in simulation. Therefore, it is useful to combine data and behavior (evaluating inegrals) together.

4.13.2 Member Function Documentation

4.13.2.1 virtual double SmartParameter::Integral (double *time1,* double *time2*) const [pure virtual]

evaluate integral at interval (time1, time2)

Parameters

time1	left range of time interval
time2	right range of time interval

Implemented in ConstParameter.

4.13.2.2 virtual double SmartParameter::IntegralSquare (double *time1,* double *time2*) const [pure virtual]

evaluate integral of square

Int(x^2)

Implemented in ConstParameter.

4.13.2.3 virtual double SmartParameter::operator() (double *t* = 0) const [pure virtual]

get value at a give time

Parameters

t	time, default value is zero

Implemented in ConstParameter.

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

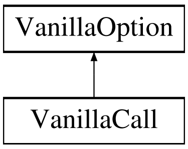
- src/lib/market_parameters.h

4.14 VanillaCall Class Reference

Vanilla call option.

```
#include <vanilla_option.h>
```

Inheritance diagram for VanillaCall:



Public Member Functions

- **VanillaCall** (double strike, double expiration)
- double PayOff (double spot) const

  pay off for a given spot
- VanillaOption $*$ clone ()

  virtual constructor

Additional Inherited Members

4.14.1 Detailed Description

Vanilla call option.

4.14.2 Member Function Documentation

4.14.2.1 double VanillaCall::PayOff (double *spot*) const [virtual]

pay off for a given spot

Parameters

spot	spot price, should be larger than 0 (>0); otherwise throw an exception.

Implements VanillaOption.

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

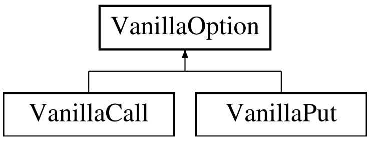
- src/lib/vanilla_option.h
- src/lib/vanilla_option.cc

4.15 VanillaOption Class Reference

vanilla option class. Interface for call or put option with a specified expiration date and strike.

`#include <vanilla_option.h>`

Inheritance diagram for VanillaOption:



Public Member Functions

- **VanillaOption** (double strike, double expiration)
- virtual double PayOff (double spot) const =0

  pay off for a given spot
- virtual VanillaOption $*$ clone ()=0

  virtual constructor

Public Attributes

- const double strike_

  strike price
- const double expiration_

  expiration date. Unit not specified

4.15.1 Detailed Description

vanilla option class. Interface for call or put option with a specified expiration date and strike.

Note

Does not specify whether it is European or American option. Can be used with either pricer.

4.15.2 Member Function Documentation

4.15.2.1 virtual double VanillaOption::PayOff (double *spot*) const `[pure virtual]`

pay off for a given spot

Parameters

spot	spot price, should be larger than 0 (>0); otherwise throw an exception.

Implemented in VanillaPut, and VanillaCall.

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

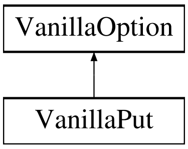
- src/lib/vanilla_option.h

4.16 VanillaPut Class Reference

Vanilla put option.

`#include <vanilla_option.h>`

Inheritance diagram for VanillaPut:



Public Member Functions

- **VanillaPut** (double strike, double expiration)
- double PayOff (double spot) const

  pay off for a given spot
- VanillaOption ∗ clone ()

  virtual constructor

Additional Inherited Members

4.16.1 Detailed Description

Vanilla put option.

4.16.2 Member Function Documentation

4.16.2.1 double VanillaPut::PayOff (double *spot*) const `[virtual]`

pay off for a given spot

Parameters

spot	spot price, should be larger than 0 (>0); otherwise throw an exception.

Implements VanillaOption.

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

- src/lib/vanilla_option.h
- src/lib/vanilla_option.cc

Chapter 5

File Documentation

5.1 src/lib/park_miller_rand_cwrapper.c File Reference

```
#include "park_miller_rand.h"
#include "rand_generator.h"
#include "park_miller_rand_cwrapper.h"
```

Functions

- int ∗ n_rand (int num, int seed)

  Generate n random numbers.
- double ∗ uniform_rand (int num, int seed)

  generate an array of uniformly distributed random number
- double ∗ norm_rand (int num, int seed)

  generate an array of normally distributed random number

5.1.1 Function Documentation

5.1.1.1 int∗ n_rand (int *num,* int *seed*)

Generate n random numbers.

Parameters

num	the number of random numbers to return
seed	seed for random number generation

Returns

an array of num of random numbers

Intends to be a wrapper to call from Python

5.1.1.2 double∗ norm_rand (int *num,* int *seed*)

generate an array of normally distributed random number

Parameters

num	number of random number to generate
seed	seed for random number generator

Returns

an array of num of random numbers

5.1.1.3 double∗ uniform_rand (int *num,* int *seed*)

generate an array of uniformly distributed random number

Parameters

num	number of random number to generate
seed	seed for random number generator

Returns

an array of num of random numbers

5.2 src/lib/park_miller_rand_cwrapper.h File Reference

A wrapper to turn C++ object into C function; intends to call from Python.

```
#include "park_miller_rand.h"
```

Functions

- int ∗ n_rand (int num, int seed)

  Generate n random numbers.
- double ∗ uniform_rand (int num, int seed)

  generate an array of uniformly distributed random number
- double ∗ norm_rand (int num, int seed)

  generate an array of normally distributed random number

5.2.1 Detailed Description

A wrapper to turn C++ object into C function; intends to call from Python.

5.2.2 Function Documentation

5.2.2.1 int∗ n_rand (int *num,* int *seed*)

Generate n random numbers.

Parameters

num	the number of random numbers to return
seed	seed for random number generation

Returns

an array of num of random numbers

Intends to be a wrapper to call from Python

5.2.2.2 double∗ norm_rand (int *num,* int *seed*)

generate an array of normally distributed random number

Parameters

num	number of random number to generate
seed	seed for random number generator

Returns

an array of num of random numbers

5.2.2.3 double∗ uniform_rand (int *num,* int *seed*)

generate an array of uniformly distributed random number

Parameters

num	number of random number to generate
seed	seed for random number generator

Returns

an array of num of random numbers

Index