# Reference della libreria per il progetto di pricing di un'opzione

Mariacristina Romano Gianmaria Enea Roat

20 aprile 2010

# Indice

# Capitolo 1

# Indice dei tipi composti

## 1.1 Gerarchia delle classi

# Capitolo 2

# Indice dei tipi composti

## 2.1 Elenco dei tipi composti

Queste sono le classi, le struct, le union e le interfacce con una loro breve descrizione:

# Capitolo 3

# Documentazione delle classi

## 3.1 Riferimenti per la classe Currency

Classe che rappresenta la valuta.

`#include <currency.hpp>`

### Membri pubblici

- Currency ()

  *costruttore di default*

- Currency (const string &c)

  *costruttore*

- ~Currency ()

  *distruttore*

- void Set_currency (const string &c)

  *funzione per impostare la valuta.*

- string Get_currency () const

  *funzione per ottenere la valuta.*

- const Currency & operator= (const Currency &p)

  *operatore di assegnamento.*

- bool operator== (const Currency &dat) const

  *operatore logico di uguaglianza.*

- bool operator!= (const Currency &dat) const

  *operatore logico di disuguaglianza.*

## Attributi privati

- int **_curr**

## Friend

- ostream & operator<< (ostream &os, const Currency &cu)

  *overloading dell'operatore "<<"*

### 3.1.1 Descrizione dettagliata

Classe di livello 0.

Classe che rappresenta la valuta. Contiene tra i membri privati un intero al quale corrisponde una valuta predefinita.

La valuta impostata di default è l'euro (EUR).

Definizione alla linea 44 del file currency.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

- currency.hpp
- currency.cpp

## 3.2 Riferimenti per la classe Deltatime

Classe che rappresenta un intervallo temporale.

`#include <deltatime.hpp>`

### Membri pubblici

- Deltatime ()
  *costruttore di default*
- Deltatime (int y, int M, int d, int h, int m, int s)
  *costruttore*
- Deltatime (const Deltatime &copy)
  *costruttore di copia*
- ~Deltatime ()
  *distruttore*
- int Get_y () const
  *funzione che restituisce l'anno*
- int Get_M () const
  *funzione che restituisce il mese*
- int Get_d () const
  *funzione che restituisce il giorno*
- int Get_h () const
  *funzione che restituisce il ora*
- int Get_m () const
  *funzione che restituisce il minuto*
- int Get_s () const
  *funzione che restituisce il secondo*
- void Set_y (int YYYY)
  *funzione che imposta l' anno*
- void Set_M (int M)
  *funzone che imposta il mese*
- void Set_d (int D)
  *funzone che imposta il giorno*
- void Set_h (int h)
  *funzone che imposta l'ora*

- void Set_m (int m)

  *funzone che imposta i minuti*

- void Set_s (int s)

  *funzone che imposta i secondi*

- double Value () const

  *valutazione del deltatime in frazione di anni*

- double operator∗ (double c) const

  *overloading dell'operatore di moltiplicazione: valuta l'oggetto in frazione di anni e lo moltiplica per un double. Restituisce il valore in frazione di anni*

- double operator/ (double c) const

  *overloading dell'operatore di divisione: valuta l'oggetto in frazione di anni e lo divide per un double. Restituisce il valore in frazione di anni*

- Deltatime operator/ (int c)

  *overloading dell'operatore di divisione per un intero*

- double operator- (double c) const

  *overloading dell'operatore di sottrazione: valuta l'oggetto in frazione di anni e sottrae un double. Restituisce il valore in frazione di anni*

- Deltatime operator- (const Deltatime &c) const

  *overloading dell'operatore di sottrazione tra oggetti Deltatime*

- const Deltatime & operator+= (const Deltatime &d)

  *overloading dell'operatore di incremento*

- double operator/ (const Deltatime &c) const

  *overloading dell'operatore di divisione tra oggetti Deltatime: valuta i 2 oggetti e restituisce il valore in frazione di anni*

- Deltatime operator+ (const Deltatime &c) const

  *overloading dell'operatore di somma*

- bool operator<= (const Deltatime &d) const

  *overloading dell'operatore logico "minore o uguale"*

- bool operator>= (const Deltatime &d) const

  *overloading dell'operatore logico "maggiore o uguale"*

- bool operator< (const Deltatime &d) const

  *overloading dell'operatore logico "minore"*

- bool operator> (const Deltatime &d) const

  *overloading dell'operatore logico "maggiore"*

- bool operator== (const Deltatime &d) const

*overloading dell'operatore logico di uguaglianza*

- bool operator!= (const Deltatime &d) const

  *overloading dell'operatore logico di disuguaglianza*

- const Deltatime & operator= (const Deltatime &copy)

  *overloading dell'operatore di assegnamento*

## Attributi privati

- int **_y**
- int **_M**
- int **_d**
- int **_h**
- int **_m**
- int **_s**

## Friend

- double operator∗ (double c, const Deltatime &del)

  *overloading dell'operatore di moltiplicazione: moltiplica un double per l'oggetto valutato in frazione di anni. Restituisce il valore in frazione di anni*

- double operator/ (double c, const Deltatime &del)

  *overloading dell'operatore di divisione: divide un double per l'oggetto valutato in frazione di anni. Restituisce il valore in frazione di anni*

- double operator- (double c, const Deltatime &del)

  *overloading dell'operatore di sottrazione: sottrae a un double l'oggetto valutato in frazione di anni. Restituisce il valore in frazione di anni*

- ostream & operator<< (ostream &os, const Deltatime &d)

  *overloading dell'operatore "<<"*

### 3.2.1 Descrizione dettagliata

Classe di livello 1.

Classe che rappresenta un intervallo temporale espresso in anni, mesi giorni, ore, minuti e secondi. Importante la ridefinizione degli operatori che permette di trattare un intervallo temporale come frazione di anni espressa da un double. Abbiamo ritenuto opportuno ridefinire in modo differente la divisione per un numero reale(double) e per un numero intero. Mentre è più comodo trattare la prima come un double che rappresenti la frazione di anni, nel secondo caso abbiamo scelto di mantenere un oggetto Deltatime.

Definizione alla linea 43 del file deltatime.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

- deltatime.hpp
- deltatime.cpp

## 3.3 Riferimenti per la classe Eq_description

Classe che rappresenta la descrizione di un'azione.

`#include <eq_description.hpp>`

### Membri pubblici

- Eq_description ()

  *costruttore di default*

- Eq_description (const string &n, const string &c, Yield_curve *cu, Volatility *v)

  *costruttore*

- Eq_description (const Eq_description &copy)

  *costruttore di copia*

- ~Eq_description ()

  *distruttore*

- void Set_name (const string &n)

  *funzione che imposta il nome dell'azione*

- void Set_code (const string &c)

  *funzione che imposta il codice dell'azione*

- void Set_curve (Yield_curve *cu)

  *funzione che imposta la curva dei tassi a cui fa riferimento l'azione*

- void Set_vol (Volatility *v)

  *funzione che imposta la curva di volatilità a cui fa riferimento l'azione*

- string Get_name () const

  *funzione che restituisce il nome dell'azione*

- string Get_code () const

  *funzione che restituisce il codice dell'azione*

- Yield_curve * Get_curve () const

  *funzione che restituisce il puntatore alla curva dei tassi a cui fa riferimento l'azione*

- Volatility * Get_vol () const

  *funzione che restituisce il puntatore alla curva di volatilità a cui fa riferimento l'azione*

- const Eq_description & operator= (const Eq_description &eq)

  *overloading dell'operatore di assegnamento*

## Attributi privati

- string **_name**
- string **_code**
- Volatility * **_vol**
- Yield_curve * **_curve**

## Friend

- std::ostream & operator<< (std::ostream &os, const Eq_description &dat)

  *overloading dell'operatore "<<"*

### 3.3.1 Descrizione dettagliata

Classe di livello 3.

Rappresenta la descrizione di un'azione.

Le sue variabili private contengono il nome dell'azione, il suo codice e i puntatori alle curve dei tassi e della volatilità a cui fa riferimento

Definizione alla linea 49 del file eq_description.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

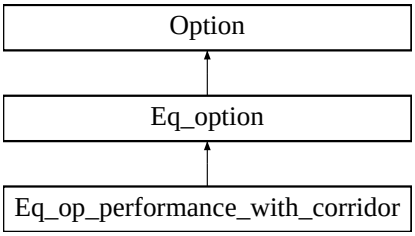
- eq_description.hpp
- eq_description.cpp

## 3.4 Riferimenti per la classe Eq_op_performance_with_corridor

Classe che rappresenta il contratto dell'opzione in esame.

`#include <eq_op_performance_with_corridor.hpp>`

Diagramma delle classi per Eq_op_performance_with_corridor:




### Membri pubblici

- Eq_op_performance_with_corridor ()
  *costruttore di default*
- Eq_op_performance_with_corridor (Eq_price eq_price, Timestruct times, double strikeprice, double lambda)
  *costruttore*
- Eq_op_performance_with_corridor (const Eq_op_performance_with_corridor &copy)
  *costruttore di copia*
- ~Eq_op_performance_with_corridor ()
  *distruttore*
- virtual Price Pay_off (Path *cammino)
  *funzione che restituisce il payoff dell'opzione ( come definito nel commento della classe) dato un cammino*
- double Get_lambda () const
  *funzione che restituisce il valore della costante lambda*
- double Get_strikeprice () const
  *funzione che restituisce lo strikeprice*
- void Set_lambda (double lambda)
  *funzione che imposta il valore della costante lambda*
- void Set_strikeprice (double strikeprice)
  *funzione che imposta lo strikeprice*
- virtual Eq_op_performance_with_corridor & operator= (Eq_op_performance_with_corridor &obj)
  *overloading dell'operatore di assegnamento*

## Attributi privati

- double **_strikeprice**
- double **_lambda**

## Friend

- ostream & operator<< (ostream &os, const Eq_op_performance_with_corridor &dat)

  *overloading dell'operatore "<<"*

### 3.4.1 Descrizione dettagliata

Classe di livello 7.

Rappresenta un particolare tipo di opzione di tipo europeo, path dependent secondo questo contratto:

Sia T la data di maturità e t=0 la data attuale, indichiamo con S(t) il prezzo del sottostante al tempo t.

Definiamo il Pay-off = 1 ∗ valuta ∗ Max[P-E,0], dove E è una sorta di strike-price dell'opzione e P è definito come la sommatoria su "i" da 1 a m dei P(i) così definiti:

P(i) = Min[ Max[ S(t+1)/S(t) - 1 , l], u]

m rappresenta il numero di intervalli regolari in cui viene suddiviso l' intervallo temporale (0,T) e definiamo dt = T / m .( (T - t(0))/m ). Definiamo l e u come:

l = EXP(r ∗ dt)/k - 1

u = EXP(r ∗ dt)∗k -1

dove k è una costante che dipende dalla grandezza dt : k = 1 + lambda ∗ sqrt(dt).

Definizione alla linea 60 del file eq_op_performance_with_corridor.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

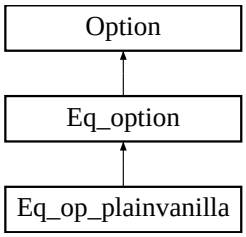
- eq_op_performance_with_corridor.hpp
- eq_op_performance_with_corridor.cpp

## 3.5 Riferimenti per la classe Eq_op_plainvanilla

Classe che rappresenta un'opzione call di tipo plain-vanilla.

`#include <eq_op_plainvanilla.hpp>`

Diagramma delle classi per Eq_op_plainvanilla:




### Membri pubblici

- Eq_op_plainvanilla ()
  *costruttore di default*
- Eq_op_plainvanilla (Eq_price eq_price, Timestruct times, double strikeprice)
  *costruttore*
- Eq_op_plainvanilla (const Eq_op_plainvanilla &copy)
  *costruttore di copia*
- ~Eq_op_plainvanilla ()
  *distruttore*
- virtual Price Pay_off (Path *cammino)
  *funzione che restituisce il payoff dell'opzione dato un cammino*
- virtual Eq_op_plainvanilla & operator= (Eq_op_plainvanilla &obj)
  *overloading dell'operatore di assegnamento*

### Attributi privati

- double **_strikeprice**

### Friend

- ostream & operator<< (ostream &os, const Eq_op_plainvanilla &dat)
  *overloading dell'operatore "<<"*

### 3.5.1 Descrizione dettagliata

Classe di livello 7.

Rappresenta un'opzione plainvanilla con strike-price E e Pay-off definito come

Pay-off = Max[P-E,0]

dove P è il prezzo del sottostante al tempo T di maturità.

Questo tipo di contratto può essere usato per fare un test della libreria in quanto per opzioni di questo tipo conosciamo le soluzioni in formula chiusa.

Definizione alla linea 53 del file eq_op_plainvanilla.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

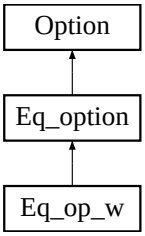
- eq_op_plainvanilla.hpp
- eq_op_plainvanilla.cpp

## 3.6 Riferimenti per la classe Eq_op_w

Classe che rappresenta un'opzione utilizzata per il controllo della libreria.

`#include <eq_op_w.hpp>`

Diagramma delle classi per Eq_op_w:




### Membri pubblici

- Eq_op_w ()
  *costruttore di default*
- Eq_op_w (Eq_price eq_price, Timestruct times)
  *costruttore*
- Eq_op_w (const Eq_op_w &copy)
  *costruttore di copia*
- ~Eq_op_w ()
  *distruttore*
- virtual Price Pay_off (Path *cammino)
  *funzione che restituisce il payoff dell'opzione dato un cammino*
- virtual Eq_op_w & operator= (Eq_op_w &obj)
  *overloading dell'operatore di assegnamento*

### Friend

- ostream & operator<< (ostream &os, const Eq_op_w &dat)
  *overloading dell'operatore "<<"*

### 3.6.1 Descrizione dettagliata

Classe di livello 7.

Rappresenta un'opzione che usiamo per testare la nostra libreria in quanto ne conosciamo a priori il risultato, inoltre è anche un test per la classe che genera numeri casuali in quanto il Pay-off è definito come la sommatoria dei quadrati delle variabili casuali estratte durante il cammino passato alla funzione Pay_off.

Definizione alla linea 50 del file eq_op_w.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

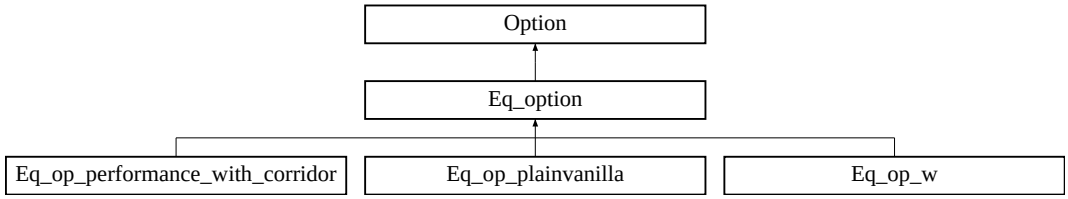
- eq_op_w.hpp
- eq_op_w.cpp

## 3.7 Riferimenti per la classe Eq_option

Classe che rappresenta il concetto di opzione.

`#include <eq_option.hpp>`

Diagramma delle classi per Eq_option:




### Membri pubblici

- Eq_option ()
  *costruttore di default*
- Eq_option (const Eq_price &eq_price, const Timestruct &times)
  *costruttore*
- Eq_option (const Eq_option &copy)
  *costruttore di copia*
- ~Eq_option ()
  *distruttore*
- virtual Price Pay_off (Path *cammino)=0
  *funzione che restituisce il pay-off du un determinato cammino*
- const Eq_price & Get_Eq_price () const
  *funzione che restituisce il punto iniziale dell'azione*
- const Timestruct & Get_times () const
  *funzione che restituisce il vettore di tempi*
- void Set_Eq_price (const Eq_price &p)
  *funzione che imposta il punto iniziale dell'azione*
- void Set_times (const Timestruct &t)
  *funzione che imposta il vettore di tempi*
- virtual Eq_option & operator= (Eq_op_performance_with_corridor &obj)
  *overloading dell'operatore di assegnamento per la classe figlia: Eq_op_performance_with_corridor*
- virtual Eq_option & operator= (Eq_op_plainvanilla &obj)
  *overloading dell'operatore di assegnamento per la classe figlia: Eq_op_plainvanilla*

- virtual Eq_option & operator= (Eq_op_w &obj)

  *overloading dell'operatore di assegnamento per la classe figlia: Eq_op_w*

## Attributi privati

- Eq_price **_eq_price**
- Timestruct **_times**

### 3.7.1 Descrizione dettagliata

Classe di livello 7.

E' la classe madre delle opzioni su un'azione.

Rappresenta una generica opzione su un sottostante e quindi la funzione Pay_off(Path) è definita virtual pura.

Definizione alla linea 54 del file eq_option.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

- eq_option.hpp
- eq_option.cpp

## 3.8 Riferimenti per la classe Eq_price

Classe che rappresenta il prezzo di un'azione in un istante fissato.

`#include <eq_price.hpp>`

### Membri pubblici

- Eq_price ()
  *costruttore di default*
- Eq_price (Eq_description *eq, Time t, Price p)
  *costruttore*
- Eq_price (Eq_description *eq)
  *costruttore con solo puntatore a equity description*
- Eq_price (const Eq_price &copy)
  *costruttore di copia*
- ~Eq_price ()
  *distruttore*
- void Set_price (const Price &p)
  *funzione che imposta il prezzo*
- void Set_eq (Eq_description *eq)
  *funzione che imposta l'anagrafica dell'azione*
- void Set_time (const Time &t)
  *funzione che imposta l'istante temporale*
- Price Get_Eq_price () const
  *funzione che restituisce il prezzo*
- Eq_description * Get_eq () const
  *funzione che restituisce l'anagrafica dell'azione*
- Time Get_time () const
  *funzione che restituisce l'istante temporale*
- Currency * Get_currency () const
  *funzione che restituisce la valuta a cui fa riferimento il prezzo*
- double Get_price_double () const
  *funzione che restituisce il valore del prezzo in double*
- const Eq_price & operator= (const Eq_price &p)
  *overloading dell'operatore di assegnamento*

## Attributi privati

- Eq_description * **_equity**
- Price **_price**
- Time **_time**

## Friend

- ostream & operator<< (ostream &os, const Eq_price &dat)

  *overloading dell'operatore "<<"*

### 3.8.1 Descrizione dettagliata

Classe di livello 4.

Classe per la gestione del valore di un'azione ad un determinato istante Time.

Contiene quindi un puntatore alla classe Eq_description che permette di identificare a quale azione fa riferimento, un membro price che contiene il valore del prezzo ed un membro Time, che indica il momento in cui l'azione assume tale prezzo.

Per comodità abbiamo implementato un metodo che restituisca direttamente il valore del prezzo in double.

Definizione alla linea 50 del file eq_price.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

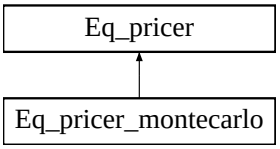
- eq_price.hpp
- eq_price.cpp

## 3.9 Riferimenti per la classe Eq_pricer

Classe che rappresenta un generico metodo di simulazione, per ottenere i prezzi di un'opzione.

`#include <eq_pricer.hpp>`

Diagramma delle classi per Eq_pricer:




### Membri pubblici

- Eq_pricer ()

  *costruttore di default*

- ~Eq_pricer ()

  *distruttore*

- virtual void Compute_price ()=0

  *funzione che valuta il prezzo dell'opzione ( virtual pura)*

- Price Get_price () const

  *funzione che restituisce il prezzo*

- Eq_option * Get_Eq_option () const

  *funzione che restituisce l'opzione di riferimento*

- Process * Get_process () const

  *funzione che restituisce il processo utilizzato*

- void Set_Eq_option (Eq_option *eq_option)

  *funzione che imposta l'opzione di riferimento*

- void Set_price (const Price &price)

  *funzione che imposta il prezzo*

- void Set_process (Process *process)

  *funzionie che imposta il processo da utilizzare*

- virtual Eq_pricer & operator= (Eq_pricer_montecarlo &obj)

  *overloading dell'operatore di assegnamento per la classe figlia: Eq_pricer_montecarlo*

## Attributi privati

- Eq_option * **_eq_option**
- Process * **_process**
- Price **_price**

### 3.9.1 Descrizione dettagliata

Classe di livello 8.

E' una classe astratta, madre delle simulazioni dei prezzi di un'opzione. Il metodo Compute_price è dichiarato virtual puro.

Consente di impostare e ottenere il prezzo e i puntatori all'opzione considerata e al processo utilizzato.

Definizione alla linea 51 del file eq_pricer.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

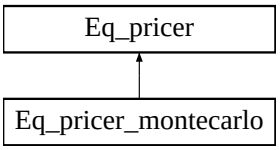
- eq_pricer.hpp
- eq_pricer.cpp

## 3.10 Riferimenti per la classe Eq_pricer_montecarlo

Classe che rappresenta il metodo di montecarlo per il pricing di un'opzione.

`#include <eq_pricer_montecarlo.hpp>`

Diagramma delle classi per Eq_pricer_montecarlo:




### Membri pubblici

- Eq_pricer_montecarlo ()
  *costruttore di default*
- Eq_pricer_montecarlo (Eq_option ∗opt, Process ∗pro, int N)
  *costruttore*
- ∼Eq_pricer_montecarlo ()
  *distruttore*
- virtual void Compute_price ()
  *valutazione del prezzo dell'opzione*
- const Stima_montecarlo & Get_Stima (const string &type) const
  *funzione che restituisce la stima del prezzo dell'opzione*
- const Statistica & Get_stat_std () const
  *funzione che restituisce la statistica del montecarlo*
- const Statistica & Get_stat_anti () const
  *funzione che restituisce la statistica del montecarlo per anticammini*
- const Statistica & Get_stat_std_anti () const
  *funzione che restituisce ls statistica del montecarlo su cammini + anticammini*
- Price Get_price_std () const
  *funzione che restituisce il prezzo valutato sui cammini standard*
- Price Get_price_anti () const
  *funzione che restituisce il prezzo valutato sugli anticammini*
- virtual Eq_pricer_montecarlo & operator= (Eq_pricer_montecarlo &obj)
  *overloading dell'operatore di assegnamento*

## Attributi privati

- int **_N**
- Statistica **_stat_std**
- Statistica **_stat_anti**
- Statistica **_stat_std_anti**
- Stima_montecarlo **_stima_std**
- Stima_montecarlo **_stima_anti**
- Stima_montecarlo **_stima_std_anti**

## Friend

- ostream & operator<< (ostream &os, Eq_pricer_montecarlo &dat)

  *overloading dell'operatore "<<"*

### 3.10.1 Descrizione dettagliata

Classe di livello 8.

Questa classe implementa il metodo Montecarlo per il pricing di un'opzione con un numero N di simulazioni passato al costruttore.

Prende come dati di input, oltre al numero di simulazioni, il tipo di opzione e il processo che si vuole utilizzare . Il metodo Compute_price avvia la simulazione, fa la stima del prezzo e lo attualizza. Utilizza la classe Statistica per fornire una stima degli errori commessi. Il membro Stima_montecarlo salva i risultati ottenuti.

Sia gli oggetti Statistica che Stima_montecarlo sono 3, per permettere all'utilizzatore di scegliere di utilizzare a scelta una stima condotta solo sui cammini standard, solo sugli anticammini o su cammini standard e anticammini.

Definizione alla linea 56 del file eq_pricer_montecarlo.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

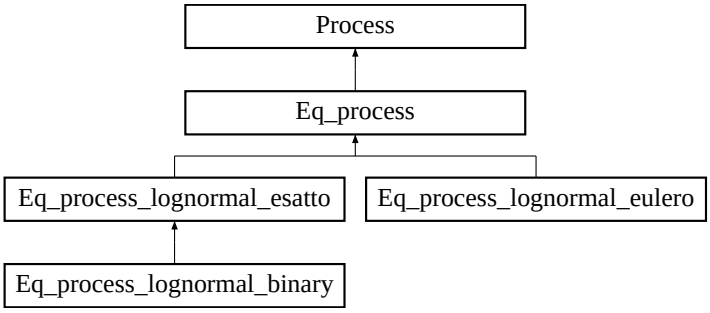
- eq_pricer_montecarlo.hpp
- eq_pricer_montecarlo.cpp

## 3.11 Riferimenti per la classe Eq_process

Classe che rappresenta il generico concetto di processo relativo ad un'azione.

`#include <eq_process.hpp>`

Diagramma delle classi per Eq_process:




### Membri pubblici

- Eq_process ()
  *costruttore di default*
- ~Eq_process ()
  *distruttore*
- virtual void Evol (Eq_price *A, Eq_price *B, N_rand *w) const =0
  *funzione che fa evolvere un'azione al tempo t1 in un'altra al tempo t2 ( virtual pura)*
- virtual N_rand * Get_w () const =0
  *funzione che restituisce il numero casuale estratto da una distribuzione ( virtual pura, l'implementazione è lasciata alle figlie)*
- virtual N_rand * Anti_w (N_rand *w) const
  *funzione che restituisce l'antitetico del numero casuale*
- Rand_gen * Get_rand_gen () const
  *funzione che restituisce il generatore di numeri casuali*
- void Set_rand_gen (Rand_gen *ra)
  *funzione che imposta il generatore di numeri casuali*
- void Set_rand_gen (void)
  *funzione che restituisce il generatore di numeri casuali di default*
- virtual Eq_process & operator= (Eq_process_lognormal_eulero &obj)
  *overloading dell'operatore di assegnamento per la classe figlia: Eq_process_lognormal_eulero*
- virtual Eq_process & operator= (Eq_process_lognormal_esatto &obj)
  *overloading dell'operatore di assegnamento per la classe figlia: Eq_process_lognormal_esatto*

## Attributi privati

- Rand_gen * **_ra**

### 3.11.1 Descrizione dettagliata

Classe di livello 5.

E' la classe madre dei processi che regolano l'evoluzione temporale (da uno stato A di partenza ad uno stato di arrivo B) di un'azione secondo funzioni dipendenti da una random variabile.

Pur rimanendo una classe prevalentemente astratta, permette di gestire i numeri casuali necessari alla funzione che regola l'evoluzione.

Permettere di scegliere il generatore di numeri casuali.

Definizione alla linea 54 del file eq_process.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

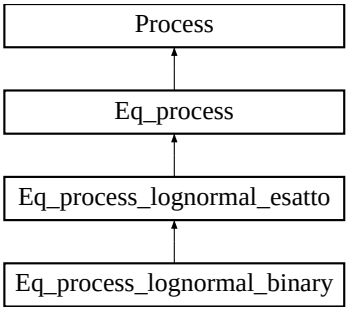
- eq_process.hpp
- eq_process.cpp

## 3.12 Riferimenti per la classe Eq_process_lognormal_binary

Classe che rappresenta un processo lognormale esatto la cui evoluzione dipende da una variabile stocastica binaria.

`#include <eq_process_lognormal_binary.hpp>`

Diagramma delle classi per Eq_process_lognormal_binary:




### Membri pubblici

- Eq_process_lognormal_binary ()
  *costruttore di default*
- Eq_process_lognormal_binary (Rand_gen ∗ra)
  *costruttore*
- ∼Eq_process_lognormal_binary ()
  *distruttore*
- virtual N_rand ∗ Get_w () const
  *funzione che restituisce un numero casuale estratto da una distribuzione binaria -1 o +1*
- virtual Eq_process_lognormal_binary & operator= (Eq_process_lognormal_binary &obj)
  *overloading dell'operatore di assegnamento*

### 3.12.1 Descrizione dettagliata

Classe di livello 5.

Rappresenta un processo lognormale esatto la cui evoluzione dipende da una random variabile estratta da una distribuzione binaria che può quindi assumere i soli valori +1 o -1. Per questo motivo è stata derivata dalla classe Eq_process_lognormal_esatto da cui eredita la formula di evoluzione, con la sola differenza che la generazione di numeri casuali viene reimplementata secondo un'estrazione binaria.

Definizione alla linea 46 del file eq_process_lognormal_binary.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

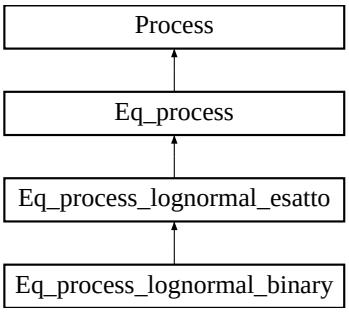
- eq_process_lognormal_binary.hpp
- eq_process_lognormal_binary.cpp

## 3.13 Riferimenti per la classe Eq_process_lognormal_esatto

Classe che rappresenta un processo lognormale esatto dipendente da una random variabile gaussiana.

`#include <eq_process_lognormal_esatto.hpp>`

Diagramma delle classi per Eq_process_lognormal_esatto:




### Membri pubblici

- Eq_process_lognormal_esatto ()
  *costruttore di default*
- Eq_process_lognormal_esatto (Rand_gen ∗ra)
  *costruttore*
- ∼Eq_process_lognormal_esatto ()
  *distruttore*
- virtual void Evol (Eq_price ∗A, Eq_price ∗B, N_rand ∗w) const
  *funzione che fa evolvere un'azione al tempo t1 in un'altra al tempo t2 secondo la funzione integrale esatta*
- virtual N_rand ∗ Get_w () const
  *funzione che restituisce il numero casuale estratto da una distribuzione gaussiana di media nulla e varianza unitaria*
- virtual Eq_process_lognormal_esatto & operator= (Eq_process_lognormal_esatto &obj)
  *overloading dell'operatore di assegnamento*
- virtual Eq_process_lognormal_esatto & operator= (Eq_process_lognormal_binary &obj)
  *overloading dell'operatore di assegnamento per la classe figlia: Eq_process_lognormal_binary*

### 3.13.1 Descrizione dettagliata

Classe di livello 5.

Rappresenta i processi che regolano l'evoluzione temporale di una azione che segua un processo lognormale secondo funzioni che dipendono da una random variabile gaussiana con media nulla e varianza unitaria. Il processo qui implementato permette la valutazione con formula integrale esatta:

P = Po∗exp( (r - (s^2)/2)∗ dt + s ∗ sqrt(dt) ∗ w )

dove P è il prezzo finale al tempo t, Po il prezzo iniziale al tempo to, r il tasso forward tra i tempi to e t, s rappresenta la volatilità, w è la variabile estratta dalla gaussiana e dt = t - to.

Definizione alla linea 49 del file eq_process_lognormal_esatto.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

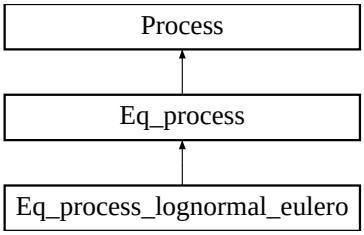
- eq_process_lognormal_esatto.hpp
- eq_process_lognormal_esatto.cpp

## 3.14 Riferimenti per la classe Eq_process_lognormal_eulero

Classe che rappresenta un processo lognormale valutato con la formula approssimata di Eulero.

`#include <eq_process_lognormal_eulero.hpp>`

Diagramma delle classi per Eq_process_lognormal_eulero:




### Membri pubblici

- Eq_process_lognormal_eulero ()
  *costruttore di default*
- Eq_process_lognormal_eulero (Rand_gen ∗ra)
  *costruttore*
- ∼Eq_process_lognormal_eulero ()
  *distruttore*
- virtual void Evol (Eq_price ∗A, Eq_price ∗B, N_rand ∗w) const
  *funzione che fa evolvere un eqprice in un altro*
- virtual N_rand ∗ Get_w () const
  *funzione che restituisce il numero casuale estratto da una distribuzione gaussiana di media nulla e varianza unitaria*
- virtual Eq_process_lognormal_eulero & operator= (Eq_process_lognormal_eulero &obj)
  *overloading dell'operatore di assegnamento*

### 3.14.1 Descrizione dettagliata

Classe di livello 5.

Rappresenta i processi che regolano l'evoluzione temporale di una azione che segua un processo lognormale secondo funzioni che dipendono da una random variabile gaussiana con media nulla e varianza unitaria. Il processo qui implementato permette la valutazione con formula approssiamta di Eulero:

P = Po + (r∗(t2-t1) + s∗(sqrt( t2 - t1 )) ∗ w) ∗ Po

dove P è il prezzo finale al tempo t2, Po il prezzo iniziale al tempo t1, r il tasso forward tra i tempi t1 e t2, s rappresenta la volatilità e w è la variabile estratta dalla gaussiana.

Definizione alla linea 47 del file eq_process_lognormal_eulero.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

- eq_process_lognormal_eulero.hpp
- eq_process_lognormal_eulero.cpp

## 3.15 Riferimenti per la classe ErrorList

Classe che rappresenta la coda di stampa degli errori.

`#include <errorlist.hpp>`

### Membri pubblici

- void Add_error (string errore)
  *funzione che consente l'aggiunta di un errore al file*
- ~ErrorList ()
  *distruttore*

### Membri pubblici statici

- static ErrorList * Get_errorlist ()
  *funzione che chiama il costruttore*

### Membri privati

- ErrorList ()
  *costruttore: è un membro privato che crea il file lista_errori.dat*

### Attributi privati

- fstream **_stampa**

### Attributi privati statici

- static ErrorList * **_list**

### 3.15.1 Descrizione dettagliata

Classe di livello 0.

Classe che rappresenta la coda di stampa degli errori di scorretto utilizzo della libreria.

Ogni volta che la libreria viene utilizzata erroneamente, viene inviato un messaggio di errore alla classe Errorlist che stamperá sul file lista_errori.dat l'elenco degli errori. Essendo un singleton, il costruttore é privato. Si evita in tal modo che venga costruito piú di un oggetto di tipo Errorlist, che sarebbe evidentemente un inutile spreco di memoria, e tutti i messaggi di errori vengono quindi gestiti da un unico oggetto.

Definizione alla linea 44 del file errorlist.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

- errorlist.hpp

- errorlist.cpp

## 3.16 Riferimenti per la classe N_rand

Questa classe rappresenta un contenitore di numeri generati casualmente.

`#include <n_rand.hpp>`

### Membri pubblici

- N_rand ()
  *costruttore di default*
- N_rand (double w)
  *costruttore*
- N_rand (const N_rand &w)
  *costruttore di copia*
- ∼N_rand ()
  *distruttore*
- const N_rand & operator= (const N_rand &n)
  *overloading dell'operatore di assegnamento*
- void Set_rand (double w)
  *funzione che imposta il numero*
- double Get_rand () const
  *funzione che restituisce il numero*
- N_rand operator- () const
  *overloading dell'operatore unario " - " che restituisce l'opposto additivo*
- double operator∗ (double a) const
  *overloading dell'operatore di moltiplicazione per un double*

### Attributi privati

- double **_w**

### Friend

- ostream & operator<< (ostream &os, const N_rand &dat)
  *overloading dell'operatore "<<"*
- double operator∗ (double a, const N_rand &num)
  *overloading dell'operatore di moltiplicazione di un double per un numero random*

### 3.16.1 Descrizione dettagliata

Classe di livello 0.

Questa classe rappresenta un contenitore di numeri generati casualmente.

Definizione alla linea 44 del file n_rand.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

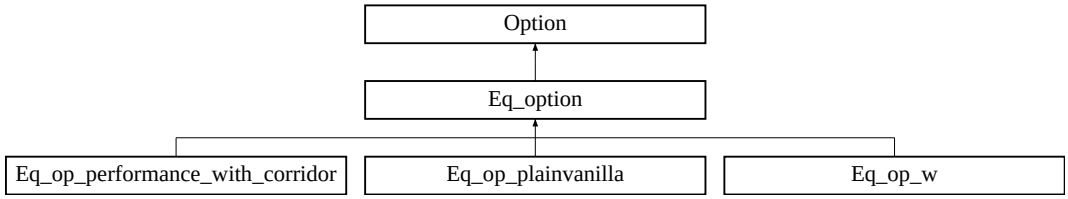
- n_rand.hpp
- n_rand.cpp

## 3.17 Riferimenti per la classe Option

Classe che rappresenta il generico concetto di opzione.

`#include <option.hpp>`

Diagramma delle classi per Option:




### Membri pubblici

- Option ()
  *costruttore di default*
- ∼Option ()
  *distruttore*
- virtual Option & operator= (Eq_option &obj)
  *overloading dell'operatore di assegnamento per la classe figlia: Eq_option.*

### 3.17.1 Descrizione dettagliata

Classe di livello 0.

Classe astratta che rappresenta genericamente un'opzione. Ogni tipo di opzione verrá ovviamente derivata da questa, implementando la funzione virtual pura definente il payoff.

L'unico metodo implementato è l'operatore di assegnamento per la classe figlia Eq_option, dichiarato virtual, necessario per permettere una maggiore elasticità del codice.

Definizione alla linea 45 del file option.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

- option.hpp
- option.cpp

## 3.18 Riferimenti per la classe Path

Classe che rappresenta il cammino di un'azione.

`#include <path.hpp>`

### Membri pubblici

- Path ()

  *costruttore di default*

- Path (const Eq_price &start_price, Timestruct *times, Process *process)

  *costruttore*

- Path (const Path &copy)

  *costruttore di copia*

- ~Path ()

  *distruttore*

- void Refresh ()

  *funzione che rigenera il cammino secondo nuovi numeri casuali, utile nella simulazione dove non è necessario salvare i path*

- void Reverse ()

  *funzione che trasforma il cammino nel suo anticammino*

- void Reverse (Path *cammino)

  *funzione che imposta l'oggetto chiamante come anticammino dell'oggetto passato alla funzione*

- Eq_price * Get_eqprice (int i)

  *funzione che restiruisce l'azione i-esima*

- int Get_dim () const

  *funzione che restituisce il vettore di tempi*

- N_rand * Get_nrand (int i) const

  *funzione che restituisce l'i-esimo numero casuale utilizzato per creare il cammino*

- const Path & operator= (const Path &cammino)

  *overloading dell'operatore di assegnamento*

- Path operator- () const

  *overloading dell'operatore unario per ottenere un anticammino*

## Attributi privati

- Eq_price ** **_eq_prices**
- N_rand ** **_num_rand**
- int **_dim**
- Process * **_process**

### 3.18.1 Descrizione dettagliata

Classe di livello 6.

Rappresenta il cammino di evoluzione di un'azione, ovvero i diversi valori che l'azione assume durante un certo lasso di tempo, controlati con una certa periodicità.

Richiamando le classi Process permette di costrutire un cammino, dato un vettore di Deltatime ( una Timestruct da cui ricaviamo anche la dimensione), a partire da un'azione a prezzo e tempo fissati.

Permette inoltre di salvare i numeri random utilizzati per generare ogni passo del cammino, e di generare l'anticammino.

Definizione alla linea 49 del file path.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

- path.hpp
- path.cpp

## 3.19 Riferimenti per la classe Price

Classe che rappresenta il prezzo.

`#include <price.hpp>`

### Membri pubblici

- Price ()
  *costruttore di default*
- Price (double c)
  *costruttore senza curruncy*
- Price (double p, Currency ∗c)
  *costruttore*
- ∼Price ()
  *distruttore*
- void Set_price (double p)
  *funzione che imposta il prezzo*
- void Set_curr (Currency ∗c)
  *funzione che imposta la valuta*
- double Get_price () const
  *funzione che restituisce il prezzo*
- Currency ∗ Get_currency () const
  *funzione che restituisce la valuta*
- const Price & operator= (const Price &p)
  *overloading dell'operatore di assegnamento*
- Price operator∗ (double p) const
  *overloading dell'operatore di moltiplicazione per un double*
- Price operator+ (double p) const
  *overloading dell'operatore di somma per un double*
- const Price & operator+= (Price p)
  *overloading dell'operatore iterativo " += "*
- Price operator/ (double d) const
  *overloading dell'operatore di divisione per un double*
- Price operator/ (int d) const
  *overloading dell'operatore di divisione per un int*

## Attributi privati

- double **_price**
- Currency * **_val**

## Friend

- ostream & operator<< (ostream &os, const Price &pri)

  *overloading dell'operatore "<<"*

- double operator/ (double d, const Price &p)

  *overloading dell'operatore di divisione di un double per un un prezzo*

### 3.19.1 Descrizione dettagliata

Classe di livello 1.

Rappresenta un prezzo, pensato come un numero reale positivo, riferito ad un'opportuna valuta.

Definizione alla linea 47 del file price.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

- price.hpp
- price.cpp

## 3.20 Riferimenti per la classe Pricing

Classe ad hoc per la risoluzione del tema d'esame.

`#include <pricing.hpp>`

### Membri pubblici

- Pricing ()

  *costruttore di default*

- Pricing (int N, double S_O, double strikeprice, double rate, double volatility, Timestruct T_maturity, double lambda, int ∗m, int dim_m, string name_process, const char ∗nome_file_out)

  *Costruttore che permette di studiare la variazione del prezzo dell'opzione per diversi valori di maturity time. La simulazione viene eseguita per ogni valore di m.*

- Pricing (int N, double S_O, double strikeprice, double rate, double volatility, const Deltatime &T_maturity, double ∗lambda, int dim_lambda, int ∗m, int dim_m, string name_process, const char ∗nome_file_out)

  *Costruttore che permette di studiare la variazione del prezzo dell'opzione per diversi valori di lambda. La simulazione viene eseguita per ogni valore di m.*

- Pricing (int ∗N, int dim_N, double S_O, double strikeprice, double rate, double volatility, const Deltatime &T_maturity, double lambda, int m, string name_process, const char ∗nome_file_out)

  *Costruttore che permette di studiare l'andamento dell'errore in dipendenza dal numero di simulazioni N.*

- Pricing (int N, double S_O, double strikeprice, double rate, double ∗volatility, int dim_vol, const Deltatime &T_maturity, double lambda, int ∗m, int dim_m, string name_process, const char ∗nome_file_out)

  *Costruttore che permette di studiare la variazione del prezzo dell'opzione per diversi valori di volatilità. La simulazione viene eseguita per ogni valore di m.*

- Pricing (int N, double S_O, double strikeprice, double rate, double volatility, Timestruct T_maturity, int ∗m, int dim_m, const char ∗nome_file_out)

  *Costruttore che permette di studiare la variazione del prezzo di un'opzione plain-vanilla. La simulazione viene eseguita per ogni valore di m.*

- Pricing (int N, double S_O, double rate, double volatility, const Deltatime &T_maturity, int ∗m, int dim_m, const char ∗nome_file_out)

  *Costruttore che permette di effettuare un test della libreria e del generatore di numeri casuali tramite l'opzione Eq_op_w.La simulazione viene eseguita per ogni valore di m.*

- ∼Pricing ()

  *distruttore*

### 3.20.1 Descrizione dettagliata

Classe di livello 9.

Classe ad hoc per la risoluzione del tema d'esame. Grazie a questa classe è possibile ottenere un file di testo (passatone il nome al costruttore) nel quale vengono stampati direttamente dalla classe i risultati della

simulazione. Al costruttore vengono passati tutti i parametri fissati (in double o int) e tutti i parametri che si vogliono far variare (come vettori di double, int o deltatime); al costruttore viene passato anche il tipo di processo (come string) che si vuole utilizzare per l'evoluzione dell'azione. Ogni costruttore rappresenta una delle simulazioni richieste dal tema d'esame.

1. Costruttore che permette di studiare la variazione del prezzo dell'opzione per diversi valori di maturity time. La simulazione viene eseguita per ogni valore di m.

2. Costruttore che permette di studiare la variazione del prezzo dell'opzione per diversi valori di lambda. La simulazione viene eseguita per ogni valore di m.

3. Costruttore che permette di studiare l'andamento dell'errore in dipendenza dal numero di simulazioni N.

4. Costruttore che permette di studiare la variazione del prezzo dell'opzione per diversi valori di volatilità. La simulazione viene eseguita per ogni valore di m.

5. Costruttore che permette di studiare la variazione del prezzo di un'opzione plain-vanilla. La simulazione viene eseguita per ogni valore di m.

6. Costruttore che permette di effettuare un test della libreria e del generatore di numeri casuali tramite l'opzione Eq_op_w.La simulazione viene eseguita per ogni valore di m.

Questa classe permette un utilizzo rapido e pratico di tutta la libreria.

Abbiamo volutamente evitato l'implementazione di un costruttore generico, implementando invece costruttori mirati ai problemi del tema d'esame, per ottimizzare l'esecuzione di ogni simulazione.

Definizione alla linea 91 del file pricing.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

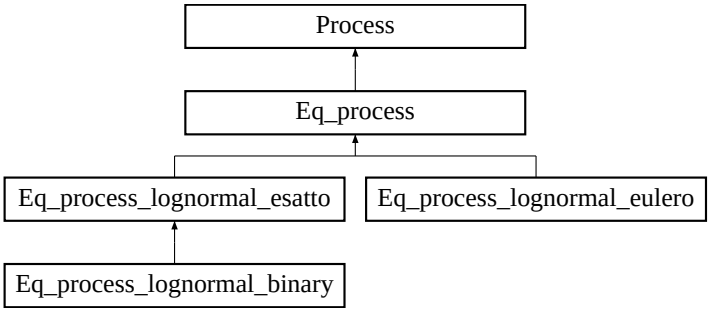
- pricing.hpp
- pricing.cpp

## 3.21 Riferimenti per la classe Process

Classe che rappresenta un generico processo.

`#include <process.hpp>`

Diagramma delle classi per Process:




### Membri pubblici

- Process ()
  *costruttore di default*
- ∼Process ()
  *distruttore*
- virtual void Evol (Eq_price ∗A, Eq_price ∗B, N_rand ∗n) const =0
  *funzione che permette ad un'azione di evolvere temporalmente*
- virtual N_rand ∗ Anti_w (N_rand ∗w) const =0
  *funzione che restituisce il valore antitetico della random variabile che le viene passata*
- virtual N_rand ∗ Get_w () const =0
  *funzione che restituisce la random variabile usata per far evolvere l'azione*
- virtual Process & operator= (Eq_process &obj)
  *overloading dell'operatore di assegnamento della classe figlia: Eq_process*

### 3.21.1 Descrizione dettagliata

Classe di livello 0.

E' la classe madre dei processi che evolvono secondo funzioni dipendenti da una random variabile.

E' una classe completamente astratta; l'unico metodo implementato è l'overloading dell'operatore di assegnamento che permette di eguagliare un oggetto di questa classe ad un oggetto della classe figlia Eq_process. Questo è dichiarato virtual perchè possa essere ridefinito nella classe figlia. Questa operazione fa sì che, qualora ad un puntatore di tipo Process venga assegnato un oggetto di tipo Eq_process, venga richiamato l'operatore di assegnamento di tipo Eq_process nonostante il puntatore sia del tipo della classe madre.

Definizione alla linea 49 del file process.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

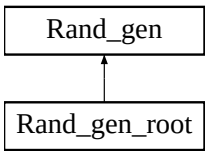
- process.hpp
- process.cpp

## 3.22 Riferimenti per la classe Rand_gen

Classe che rappresenta un generico generatore di numeri casuali.

`#include <rand_gen.hpp>`

Diagramma delle classi per Rand_gen:




### Membri pubblici

- Rand_gen ()

  *costruttore di default*

- ∼Rand_gen ()

  *distruttore*

- virtual N_rand ∗ Get_gauss (double m, double var)=0

  *funzione che restituisce un numero random estratto da una distribuzione gaussiana con media m e deviazione standard var*

- virtual N_rand ∗ Get_unif (double in, double en)=0

  *funzione che restituisce un numero random estratto da una distribuzione uniforme sull'intervallo di estremi in , en*

- virtual N_rand ∗ Get_binary ()=0

  *funzione che restituisce un numero random estratto da una distribuzione binaria*

- virtual Rand_gen & operator= (Rand_gen_root &obj)

  *overloading dell'operatore di assegnamento per la classe figlia: Rand_gen_root*

### 3.22.1 Descrizione dettagliata

Classe di livello 1.

E' la classe madre delle classi generatrici di numeri casuali.

E' una classe puramente astratta; l'unico metodo implementato è l'overloading dell'operatore di assegnamento per la classe figlia Rand_gen_root.

Possiede 3 diversi metodi per la generazione di numeri casuali, dichiarati virtual puri perchè possano essere implementati nelle diverse classi figlie:

Get_gauss restituisce un numero casuale estratto da una distribuzione gaussiana,

Get_unif che permette di estrarlo da una distribuzione uniforme,

Get_binary che permette di estrarlo da una distribuzione binaria.

Definizione alla linea 51 del file rand_gen.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

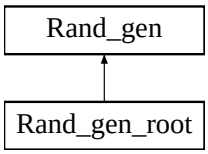
- rand_gen.hpp
- rand_gen.cpp

## 3.23 Riferimenti per la classe Rand_gen_root

Generatore di numeri casuali basato sulla libreria ROOT (CERN).

`#include <rand_gen_root.hpp>`

Diagramma delle classi per Rand_gen_root:




### Membri pubblici

- ∼Rand_gen_root ()

  *distruttore*

- virtual N_rand ∗ Get_gauss (double m, double sig)

  *funzione che permette di ottenere un numero casuale estratto da una distribuzione gaussiana con media m e deviazione standard sig*

- virtual N_rand ∗ Get_unif (double in, double en)

  *funzione che permette di ottenere un numero casuale estratto da una distribuzione uniforme nell'intervallo [in , en]*

- virtual N_rand ∗ Get_binary ()

  *funzione che permette di ottenere un numero casuale che può assumere con la stessa probabilità i valori -1 o +1*

### Membri pubblici statici

- static Rand_gen_root ∗ Get_rand_gen_root ()

  *funzione che consente di generare e ottenere il membro statico*

### Membri privati

- Rand_gen_root ()

  *costruttore di default*

### Attributi privati

- TRandom ∗ **_gen**

## Attributi privati statici

- static Rand_gen_root * **_single**

### 3.23.1 Descrizione dettagliata

Classe di livello 1.

E' una classe in grado di generare numeri casuali secondo distribuzioni gaussiana, binaria, e uniforme basata sulla classe TRandom3 della libreria ROOT (CERN).

Abbiamo scelto di formulare anche questa classe come singleton, servendo nella pratica un unico generatore di numeri casuali, ottimizzando l'utilizzo della libreria.

Definizione alla linea 47 del file rand_gen_root.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

- rand_gen_root.hpp
- rand_gen_root.cpp

## 3.24 Riferimenti per la classe Statistica

Classe che rappresenta la statistica.

`#include <statistica.hpp>`

### Membri pubblici

- Statistica ()

  *costruttore di default*

- Statistica (const Statistica &copy)

  *costruttore di copia*

- ~Statistica ()

  *distruttore*

- double Get_media () const

  *funzione che restituisce la media*

- double Get_media2 () const

  *funzione che restituisce la media dei quadrati*

- double Get_var2 () const

  *funzione che restituisce la varianza : media(x^2) - media(x)^2*

- double Get_standard_deviation () const

  *funzione che restituisce la deviazione standard*

- double **Get_errore** () const
- double Get_errore_percentuale () const

  *funzione che restituisce l'errore relativo*

- void Add_observation (double x)

  *funzione che salva un'osservazione: salva la somma delle osservabili, la somma dei loro quadrati e incrementa il counter interno*

- const Statistica & operator= (const Statistica &copy)

  *overloading dell'operatore di assegnamento*

### Attributi privati

- double **_sum**
- double **_sum2**
- int **_cont**

### 3.24.1 Descrizione dettagliata

Classe di livello 0.

Questa classe rappresenta la statistica. Conserva la somma delle osservazioni, dei loro quadrati e conserva un counter interno per poterne poi fare la media e ottenere quindi informazioni sugli errori quali varianza, deviazione standard, errore assoluto ed errore relativo.

Definizione alla linea 37 del file statistica.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

- statistica.hpp
- statistica.cpp

## 3.25 Riferimenti per la classe Stima_montecarlo

Classe che rappresenta la stima ottenuta da N simulazioni di montecarlo.

`#include <stima_montecarlo.hpp>`

### Membri pubblici

- Stima_montecarlo ()
  *costruttore di default*
- Stima_montecarlo (double prezzo, double err_assoluto)
  *costruttore*
- ~Stima_montecarlo ()
  *distruttore*
- double Get_prezzo () const
  *funzione che restituisce il prezzo calcolato*
- double Get_errore_assoluto () const
  *funzione che restituisce l'errore assoluto*
- double Get_errore_relativo () const
  *funzione che restituisce l'errore relativo*
- void Set_prezzo (double prezzo)
  *funzione che imposta il prezzo*
- void Set_errore_assoluto (double err_assoluto)
  *funzione che imposta l'errore assoluto*
- void Set (const Statistica &stat)
  *funzione che imposta prezzo ed errore, a partire da un oggetto di tipo Statistica*
- Stima_montecarlo & operator= (Stima_montecarlo &obj)
  *overloading dell'operatore di assegnamento*

### Attributi privati

- double **_prezzo**
- double **_err_assoluto**

### Friend

- ostream & operator<< (ostream &os, Stima_montecarlo &dat)
  *overloading dell'operatore "<<"*

### 3.25.1 Descrizione dettagliata

Classe di livello 1.

Rappresenta la stima ottenuta da N simulazioni di montecarlo. Salva i dati del prezzo e dell'errore assoluto in modo che siano facilmente accessibili al termine di una simulazione.

Definizione alla linea 45 del file stima_montecarlo.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

- stima_montecarlo.hpp
- stima_montecarlo.cpp

## 3.26 Riferimenti per la classe Time

Classe che rappresenta un istante temporale.

`#include <time.hpp>`

### Membri pubblici

- Time ()
  *costruttore di default*
- Time (int YYYY, int M, int D, int h, int m, int s)
  *costruttore*
- Time (const Time &copia)
  *costruttore di copia*
- ~Time ()
  *distruttore*
- int Get_year () const
  *funzione che restituisce l'anno*
- int Get_month () const
  *funzione che restituisce il mese*
- int Get_day () const
  *funzione che restituisce il giorno*
- int Get_hour () const
  *funzione che restituisce le ore*
- int Get_minute () const
  *funzione che restituisce i minuti*
- int Get_second () const
  *funzione che restituisce i secondi*
- void Set_year (int YYYY)
  *funzione che imposta l'anno ( se 29/2 e anno non bisestile -> 28//2)*
- void Set_month (int M)
  *funzione che imposta il mese ( se il giorno preimpostato non è presente nel nuovo mese, viene impostato di default l'ultimo giorno possibile)*
- void Set_day (int D)
  *funzione che imposta il giorno*
- void Set_hour (int h)
  *funzione che imposta le ore*

- void Set_minute (int m)

  *funzione che imposta i minuti*

- void Set_second (int s)

  *funzione che imposta i secondi*

- int Num_days_M (int mese) const

  *funzione che restituisce il numero di giorni del mese passato al metodo*

- int Delta_days_0 () const

  *funzione che restituisce i giorni dal primo gennaio dell'anno 0*

- int Delta_days_1 () const

  *funzione che restituisce i giorni dal primo gennaio dell'anno impostato nell'oggetto chiamante*

- int Delta_days (const Time &da) const

  *funzione che restituisce la differenza di 2 time in giorni*

- double Delta_months (const Time &da) const

  *funzione che restituisce la differenza di 2 time in frazione di mesi*

- double Delta_years (const Time &da) const

  *funzione che restituisce la differenza di 2 time in frazione di anni*

- void Plus (int pl, const string &a)

  *funzione che permette di sommare ad un time secondi, minuti, ore, giorni, mesi o anni in base a quanto specificato nella stringa*

- void Minus (int pl, const string &a)

  *funzione che permette di sottrarre a un time secondi, minuti, ore, giorni, mesi o anni in base a quanto specificato nella string*

- Time operator- (int days) const

  *overloading dell'operatore di sottrazione per numero intero di giorni*

- Time operator+ (int days) const

  *overloading dell'operatore di somma per numero intero di giorni*

- Time & operator+= (int days)

  *overloading dell'operatore iterativo di somma "+="*

- Time & operator-= (int days)

  *overloading dell'operatore iterativo di sottrazione "-="*

- Deltatime operator- (const Time &d1) const

  *overloading dell'operatore di sottrazione tra 2 Time*

- Time operator- (const Deltatime &c) const

  *overloading dell'operatore di sottrazione per un oggetto Deltatime*

- Time operator+ (const Deltatime &delta) const

  *overloading dell'operatore di somma per un Deltatime*

- Time & operator+= (const Deltatime &delta)

  *overloading dell'operatore iterativo di somma "+=" ( per Deltatime)*

- Time & operator-= (const Deltatime &delta)

  *overloading dell'operatore iterativo di sottrazione "-=" ( per Deltatime)*

- const Time & operator= (const Time &data)

  *overloading dell'operatore di assegnamento*

- bool operator== (const Time &dat) const

  *overloading dell'operatore logico di uguaglianza*

- bool operator!= (const Time &dat) const

  *overloading dell'operatore logico di disuguaglianza*

- bool operator< (const Time &dat) const

  *overloading dell'operatore logico <*

- bool operator> (const Time &dat) const

  *overloading dell'operatore logico >*

- bool operator<= (const Time &dat) const

  *overloading dell'operatore logico <=*

- bool operator>= (const Time &dat) const

  *overloading dell'operatore logico >=*

## Membri pubblici statici

- static void Set_today (const Time &today)

  *funzione che imposta l'oggeto today come copia di un oggetto Time passatogli*

- static void Set_today (int Y, int M, int D, int h, int m, int s)

  *funzione che imposta l'oggetto statico today*

- static Time * Get_today ()

  *funzioine che crea il link statico all'oggetto today*

- static void Print_today ()

  *funzione che stampa a video l'oggetto today*

## Attributi privati

- int **_YYYY**
- int **_M**
- int **_D**
- int **_h**
- int **_m**
- int **_s**

## Attributi privati statici

- static Time * **_today**

## Friend

- ostream & operator<< (ostream &os, const Time &dat)

  *overloading dell'operatore "<<"*

### 3.26.1 Descrizione dettagliata

Classe di livello 1.

Classe che rappresenta un istante temporale, quindi una data e un'orario.

Abbiamo trattato questa classe con particolare cura, implementando funzioni di controllo per gli anni bisestili, metodi e operatori di interazione con le altri classi temporali (in particolare la Deltatime), prestando attenzione al differente numero di giorni dei mesi.

La data è rappresentata come : anno, mese, giorno, ore, minuti, secondi.

Possiede un membro statico "today" che indica il giorno di "oggi". Questo singleton ci permette di gestire al meglio alcune funzioni all'interno della libreria:

se in una funzione è necessario conoscere una data si potrà anche passare un oggetto Deltatime che automaticamente, facendo riferimento a today, ricaverà la data in questione.

Ecco il metodo per stabilire se l'anno sia o meno bisestile:

bisestile: if((_YYYY4==0 && _YYYY100!=0)||(_YYYY400==0))

ovvero se l'anno è divisibile per 4, ma non per cento, oppure se è divisibile per 400.

non bisestile: if(_YYYY400!=0){if((_YYYY4!=0)||(_YYYY100==0));}

ovvero se non è divisibile per 400 e, o non è divisibile per 4 o non è divisibile per 100.

Definizione alla linea 65 del file time.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

- time.hpp
- time.cpp

## 3.27 Riferimenti per la classe Timestruct

Classe che rappresenta un vettore di intervalli temporali.

`#include <timestruct.hpp>`

### Membri pubblici

- Timestruct ()
  *costruttore di default*
- Timestruct (int dim)
  *costruttore con la sola dimensione*
- Timestruct (Deltatime *delta_tempi, int dim)
  *costruttore con vettore di deltatime*
- Timestruct (Deltatime **delta_tempi, int dim)
  *costruttore con vettore di puntatori di deltatime*
- Timestruct (const Deltatime &delta, int dim)
  *costruttore con un solo deltatime (vettore di intervalli di eguale durata)*
- Timestruct (Time *tempi, int dim)
  *costruttore con vettore di istanti (il deltatime sarà la differenza di istanti a partire da "today")*
- Timestruct (const Timestruct &copy)
  *costruttore di copia*
- ~Timestruct ()
  *distruttore*
- const Deltatime & Get_deltatime (int i)
  *funzione che restituisce l'i-esimo intervallo*
- int Get_dim () const
  *funzione che restituisce la dimensione del vettore*
- void Set_deltatime (const Deltatime &delta, int i)
  *funzione che imposta l'i-esimo intervallo*
- const Timestruct & operator= (const Timestruct &copy)
  *overloading dell'operatore di assegnamento*
- bool operator== (const Timestruct &dat) const
  *overloading dell'operatore logico di eguaglianza " == "*
- bool operator!= (const Timestruct &dat) const
  *overloading dell'operatore logico di diseguaglianza " != "*

## Attributi privati

- Deltatime * **_delta_tempi**
- int **_dim**

## Friend

- ostream & operator<< (ostream &os, const Timestruct &dat)

  *overloading dell'operatore "<<"*

### 3.27.1 Descrizione dettagliata

Classe di livello 2.

Questa classe gestisce vettori di intervalli temporali Deltatime, per cui contiene un membro che rappresenta la lunghezza di tali vettori.

Rappresenta una struttura temporale, permette una migliore e piú sicura gestione dei vettori di intervalli di tempi che verranno ampliamente usati all'interno della libreria.

Abbiamo implementato un costruttore che permetta di creare un oggetto Timestruct composto da intervalli di eguale durata passando semplicemente un Deltatime e la lughezza del vettore.

Un altro costruttore permette di passare istanti temporali: gli intervalli cengono ricavati dalla differenza di tali date e , più precisamente, il primo intervallo corrisponde alla differenza tra il primo istante passato alla funzione e "today".

Definizione alla linea 51 del file timestruct.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

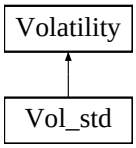
- timestruct.hpp
- timestruct.cpp

## 3.28 Riferimenti per la classe Vol_std

Classe che rappresenta una curva di volatilità costante.

`#include <vol_std.hpp>`

Diagramma delle classi per Vol_std:




### Membri pubblici

- Vol_std ()

  *costruttore di default*

- Vol_std (double v)

  *costruttore*

- Vol_std (const Vol_std &vo)

  *costruttore di copia*

- ~Vol_std ()

  *distruttore*

- void Set_vol (double v)

  *funzione che imposta il valore della volatilità*

- virtual double Get_vol (const Time &t1, const Time &t2) const

  *funzione che restituisce la volatilità tra t1 e t2*

- virtual Vol_std & operator= (Vol_std &obj)

  *overloading dell'operatore di assegnamento*

### Attributi privati

- double **_vola**

### Friend

- std::ostream & operator<< (std::ostream &os, const Vol_std &dat)

  *overloading dell'operatore "<<"*

### 3.28.1 Descrizione dettagliata

Classe di livello 2.

Rappresenta una curva di volatilità piatta, ovvero costante nel tempo. Per questo motivo è sufficiente un double per impostare la curva.

Definizione alla linea 45 del file vol_std.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

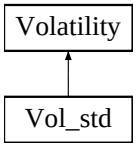
- vol_std.hpp
- vol_std.cpp

## 3.29 Riferimenti per la classe Volatility

Classe che rappresenta il concetto di curva di volatilità di un'azione.

`#include <volatility.hpp>`

Diagramma delle classi per Volatility:




### Membri pubblici

- Volatility ()
  *costruttore di default*
- ~Volatility ()
  *distruttore*
- virtual double Get_vol (const Time &t1, const Time &t2) const =0
  *funzione che restituisce la volatilità tra i tempi t1 e t2 (virtual pura)*
- virtual Volatility & operator= (Vol_std &obj)
  *operatore di assegnamento per la classe figlia: Vol_std*

### 3.29.1 Descrizione dettagliata

Classe di livello 2.

Classe astratta che traduce il concetto di volatilitá di un'azione, è quindi la classe madre delle curve di volatilità.

E' una classe puramente astratta in cui l'unico metodo implementato è l'overloading dell'operatore di assegnamento per la classe figlia.

La sua funzione principale (Get_vol) viene implementata a partire dal concetto temporale espresso dalla classe Time.

Definizione alla linea 48 del file volatility.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

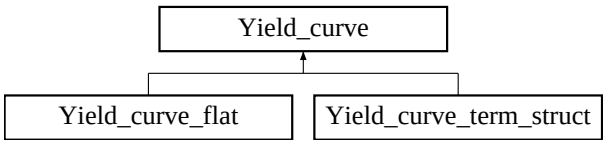
- volatility.hpp
- volatility.cpp

## 3.30 Riferimenti per la classe Yield_curve

Classe che rappresenta una generica curva dei tassi.

`#include <yield_curve.hpp>`

Diagramma delle classi per Yield_curve:




### Membri pubblici

- virtual double Get_spot_rate (const Time &t) const =0

  *restitisce il tasso ad un tempo t ( è virtual pura)*

- virtual double Get_spot_rate (const Deltatime &t) const =0

  *restitisce il tasso dopo un intervallo di tempo t ( è virtual pura)*

- double Get_discount_factor (const Time &t) const

  *funzione che restituisce il fattore di sconto ad un tempo t fissato. Il tempo deve essere positivo(= maggiore di oggi)*

- double Get_discount_factor (const Time &t1, const Time &t2) const

  *funzione che restituisce il fattore di sconto tra 2 tempi t1 e t2*

- double Get_discount_factor (const Deltatime &delta) const

  *funzione che restituisce il fattore di sconto su un intervallo temporale delta*

- double Get_forward_rate (const Time &t1, const Time &t2) const

  *funzione che restituisce il tasso forward tra i tempi t1 e t2*

- double Get_forward_rate (const Time &t, const Deltatime &delta) const

  *funzione che restituisce il tasso forward su un intervallo delta che parte dal tempo t1*

- virtual Yield_curve & operator= (Yield_curve_flat &obj)

  *overloading dell'operatore di assegnamento per la classe figlia: Yield_curve_flat*

- virtual Yield_curve & operator= (Yield_curve_term_struct &obj)

  *overloading dell'operatore di assegnamento per la classe figlia: Yield_curve_term_struct*

### 3.30.1 Descrizione dettagliata

Classe di livello 2.

E' la classe madre delle curve del tasso e per questo contiene gli overloading delle funzioni di assegnamento delle classi figlie, dichiarati virtual.

Le funzioni che restituiscono lo spot-rate ovvero il tasso ad un tempo ben preciso sono chiaramente virtual pure, poichè saranno implementate diversamente in ogni classe figlia.

Sono già implementate le funzioni Get_forward_rate e Get_discount_factor che prescindono dalla struttura delle calssi derivate poichè necessitano delle generiche funzioni Get_spot_rate.

Definizione alla linea 51 del file yield_curve.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

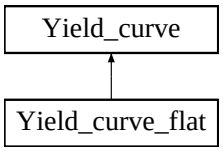
- yield_curve.hpp
- yield_curve.cpp

## 3.31 Riferimenti per la classe Yield_curve_flat

Classe che rappresenta una curva dei tassi costante.

`#include <yield_curve_flat.hpp>`

Diagramma delle classi per Yield_curve_flat:




### Membri pubblici

- Yield_curve_flat ()
  *costruttore di defalut*
- Yield_curve_flat (double rate)
  *costruttore*
- Yield_curve_flat (const Yield_curve_flat &fla)
  *costruttore di copia*
- ~Yield_curve_flat ()
  *distruttore*
- virtual double Get_spot_rate (const Time &t) const
  *funzione che restituisce il valore del tasso al tempo t*
- virtual double Get_spot_rate (const Deltatime &t) const
  *funzione che restituisce il valore del tasso dopo un intervallo di tempo t*
- double Get_rate () const
  *funzione che restituisce il tasso*
- void Set_rate (double rate)
  *funzione che imposta il tasso*
- virtual Yield_curve_flat & operator= (Yield_curve_flat &obj)
  *overloading dell'operatore di assegnamento*

### Membri pubblici statici

- static void Set_riskfree (Yield_curve_flat &y)
  *funzione che imposta il tasso della curva risk-free (come copia di un altro oggetto Yield_curve_flat)*
- static void Set_riskfree (double rate)

*funzione che imposta il tasso della curva risk-free (dato un double)*

- static Yield_curve_flat * Get_riskfree ()

  *funzione che restituisce il tasso risk-free (statico)*

## Attributi privati

- double **_rate**

## Attributi privati statici

- static Yield_curve_flat * **_riskfree**

## Friend

- std::ostream & operator<< (std::ostream &os, const Yield_curve_flat &fla)

  *overloading dell'operatore "<<"*

### 3.31.1 Descrizione dettagliata

Classe di livello 2.

Rappresenta una curva dei tassi piatta, ovvero costante nel tempo. Contiene un membro statico per la dichiarazione della curva risk-free.

Definizione alla linea 48 del file yield_curve_flat.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

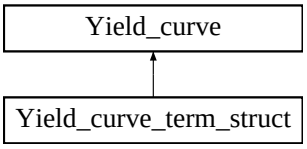
- yield_curve_flat.hpp
- yield_curve_flat.cpp

## 3.32 Riferimenti per la classe Yield_curve_term_struct

Classe che rappresenta una curva dei tassi complessa, costruita sull'interpolazione di dati.

`#include <yield_curve_term_struct.hpp>`

Diagramma delle classi per Yield_curve_term_struct:




### Membri pubblici

- Yield_curve_term_struct ()
  *costruttore di default*
- Yield_curve_term_struct (int n, Deltatime *time, double *tax)
  *costruttore (lunghezza vettori, vettore Deltatime, vettore double (tassi))*
- Yield_curve_term_struct (int n)
  *costruttore con il solo contatore*
- Yield_curve_term_struct (const Yield_curve_term_struct &nor)
  *costruttore di copia*
- ~Yield_curve_term_struct ()
  *distruttore*
- virtual Yield_curve_term_struct & operator= (Yield_curve_term_struct &nor)
  *overloading dell'operatore di assegnamento*
- virtual double Get_spot_rate (const Time &t) const
  *funzione che restituisce il tasso corrispondente al tempo t interpolando linearmente*
- virtual double Get_spot_rate (const Deltatime &t) const
  *funzione che restituisce il tasso corrispondente all'intervallo temporale t interpolando linearmente*
- int Get_n () const
  *funzione che restituisce il numero di punti*
- double Get_tax (int i) const
  *funzione che restituisce l'iesimo tasso-spot ( tasso all'i-esimo tempo)*
- Deltatime Get_time (int i) const
  *funzione che restituisce l'iesimo intervallo temporale*
- void Set_n (int n)

*funzione che imposta la dimensione, ovvero il numero di punti*

- void Set_tax (int i, double tax)

  *funzione che imposta il tasso all'i-esimo intervallo temporale*

- void Set_time (int i, const Deltatime &time)

  *funzione che imposta l'i-esimo intervallo temporale*

## Membri pubblici statici

- static void Set_riskfree (Yield_curve_term_struct &y)

  *permette di impostare la curva dei tassi risk-free ( come copia di un altro oggetto Yield_curve_term_struct)*

- static void Set_riskfree (int n, Deltatime *time, double *tax)

  *permette di impostare la curva dei tassi risk-free (con vettori)*

- static Yield_curve_term_struct * Get_riskfree ()

  *funzione che restituisce un puntatore al membro statico risk-free (e di crearlo)*

## Attributi privati

- int **_n**
- Deltatime * **_time**
- double * **_tax**

## Attributi privati statici

- static Yield_curve_term_struct * **_riskfree**

## Friend

- std::ostream & operator<< (std::ostream &os, const Yield_curve_term_struct &nor)

  *overloading dell'operatore "<<"*

### 3.32.1 Descrizione dettagliata

Classe di livello 2.

Rappresenta una curva dei tassi complessa, costruita sull'interpolazione di dati.

Supponendo di essere a conoscenza del valore del tasso a certi tempi fissati si costrutisce una curva discreta. Nei metodi da noi implemntati abbiamo utilizzato una semplice interpolazione lineare per ricavare il valore del tasso a tempi diversi da quelli "conosciuti" dall'oggetto in esame. La formula utilizzata è:

tax(t)=tax(t1)+ (tax(t2)-tax(t1)) * (t-t1) / (t2-t1)

dove t è il tempo a cui vogliamo trovare il tasso tax(t), t1 e t2 sono gli estremi dell'intervallo temporale a noi noto che contiene t e tax(t1), tax(t2) sono i tassi( a noi noti) in tali istanti.

Abbiamo considerato la curva come costante nel caso in cui t fosse compreso tra "today" e la prima data a nostra disposizione e nel caso in cui t fosse superiore all'ultima data conosciuta.

Contiene un membro statico per la curva dei tassi risk-free.

Il costruttore completo della classe accetta quindi il numero di punti conosciuti, un vettore di Deltatime( per le date di riferimento) ed un vettore di double per i tassi.

Definizione alla linea 54 del file yield_curve_term_struct.hpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

- yield_curve_term_struct.hpp
- yield_curve_term_struct.cpp

# Indice analitico