Elementi di Finanza: modelli e metodi numerici.

Marco Airoldi

E–mail:
MarcoAiroldi@yahoo.it

Versione documento 8.01

14 marzo 2012

Indice

Introduzione

Le presenti note si articolano in tre parti principali:

- la parte di finanza matematica dedicata ai modelli e ai metodi numerici (sezioni 1, 2 e 3);
- il laboratorio (sezione 4), con una serie di esercizi dedicati all'implementazione e alla prova pratica di quanto visto nella sezione teorica;
- una parte di programmazione ad oggetti in C++ (sezione 5), dove vengono richiamati e definiti i concetti fondamentali. Tale capitolo può essere letto in maniera independente dal resto delle note o può servire come base di partenza per comprendere lo sviluppo della libreria di "pricing" descritta nella sezione 3.

Parte I: matematica finanziaria

Le sezioni dedicate alla finanza quantitativa, sono nate da una serie di lezioni tenute dal sottoscritto presso il:

- corso di "Metodi Numerici - Sottomodulo II: Alberi binomiali e differenze finite" - "Master in metodologie e modelli per la finanza quantitativa". Università di Milano (Dipartimento di Fisica);
- sottomodulo di "Metodi numerici e calcolo stocastico: esempi di applicazione" del corso di "Metodi Computazionali della Fisica". Università di Milano (Dipartimento di Fisica).
- modulo di "Pricing dei derivati". Università del Piemonte Orientale (Dipartimento di economia).
- modulo di "Metodi numerici: Monte Carlo per il pricing di derivati equity". Corso di Alta Formazione in Finanza Quantitativa. MIP.

Il materiale presentato è comunque ben più ampio di quanto sarebbe richiesto per un corso di metodi numerici applicati alla finanza ed è inteso a dare allo studente una visione globale ed esaustiva dei concetti base della

finanza quantitativa e dei principali metodi numerici utilizzati per il "pricing" dei derivati (con particolare enfasi a quelli su equity).

Finalità
La finalità delle presenti note è quella di dimostrare come alcuni metodi sviluppati all'interno della fisica abbiano una vasta gamma di applicazioni anche in ambiti apparentemente lontani, quali la finanza. In particolare, le note si focalizzano proprio nell'applicazioni del calcolo stocastico e di alcuni metodi numerici al *pricing* di opzioni finanziarie esotiche [1].
Lo studio delle presenti note, insieme alla parte di laboratorio, consentirà allo studente di acquisire una buona conoscenza sia dei fondamenti della finanza quantitativa (funzionamento dei titoli derivati più comunemente trattati sul mercato, processi stocastici ecc.) sia dei principali metodi numerici utilizzati nella determinazione del prezzo dei derivati e sulla loro implementazione in una libreria finanziaria. Su quest'ultimo punto verrà posta particolare enfasi in quanto l'aspetto cruciale nell'attività di un quantitativo consiste spesso nell'implementazione pratica di algoritmi e tecniche numeriche.

Conoscenze necessarie
Si presuppone, nei partecipanti, una conoscenza matematica adeguata ai temi trattati, nonché una conoscenza di base dei fondamenti della programmazione ad oggetti, con particolare attenzione al C++.

Struttura delle sezioni (finanza quantitativa)
I capitoli principali sono strutturati nel seguente modo:

- Una parte teorica (sezione 1) il cui scopo è quello di introdurre i concetti teorici fondamentali della finanza (tassi di interesse, definizione di opzione, processi stocastici, formula di Black & Scholes, ecc.).

- Una parte teorica sul calcolo numerico (sezione 2), in cui verranno presentate a livello teorico le principali metodologie numeriche utilizzate nel campo della finanza per il calcolo del prezzo di opzione esotiche (metodo Monte Carlo, alberi binomiali e differenze finite).

- Una parte applicativa (sezione 3) in cui viene discusso in dettagli la costruzione di una libreria finanziaria tramite un linguaggio di programmazione ad oggetti.
  La lettura di questo capitolo richiede comunque una certa conoscenza dei fondamenti della programmazione ad oggetti ed andrebbe quindi preceduta da un'adeguata formazione sul tema.

[1] Uno degli algoritmi numerici su cui si concentrerà la nostra attenzione, il metodo Monte Carlo, è largamente utilizzato ed applicato in ambito fisico.

Bibliografia
Alla fine delle note è riportata una breve bibliografia (sia di articoli recentemente pubblicati su riviste, sia di libri di testo) che può essere utilizzata per approfondire gli argomenti trattati in queste note. In particolare un buon testo di approfondimento sulla finanza, con un deciso taglio economico/finanziario ed un'ottima descrizione degli stumenti finanziari principali è il volume di Hull [1]. Un testo più matematico e quantitativo è l'ottimo volume di Wilmott [2], disponibile anche in una versione più introduttiva [3]. Una presentazione delle tematiche riguardanti l'econofisica (ovvero l'applicazione di alcune tecniche e modelli della fisica all'economia) la si può trovare in [4].

Parte II: laboratorio sul "pricing" dei derivati

Come già menzionato, la sezione dedicata al laboratorio di calcolo numerico è nata all'interno di un ciclo di lezioni tenute dal sottoscritto presso il:

- corso di "Metodi Numerici - Sottomodulo II: Alberi binomiali e differenze finite" - "Master in metodologie e modelli per la finanza quantitativa". Università di Milano (Dipartimento di Fisica);
- sottomodulo di "Metodi numerici e calcolo stocastico: esempi di applicazione" del corso di "Metodi Computazionali della Fisica". Università di Milano (Dipartimento di Fisica).
- modulo di "Pricing dei derivati". Università del Piemonte Orientale (Dipartimento di economia).

Conoscenze necessarie
Si presuppone, nei partecipanti, una conoscenza di base dei fondamenti della programmazione procedurale o della programmazione ad oggetti.

Struttura della sezione dedicata alle prove di laboratorio numerico
Il contenuto dei capitoli sulla parte di laboratorio è il seguente:

- Una sessione di laboratorio (sezione 4) dove vengono proposti diversi esempi ed esercitazioni pratiche al fine di testare su casi concreti quanto presentato nelle sezioni teoriche.

Parte III: elementi di programmazione ad oggetti

La sezione dedicata alla programmazione ad oggetti in C++ è nata da una serie di lezioni tenute dal sottoscritto all'interno del:

- sottomodulo di "Programmazione ad oggetti in C++" del corso di "Metodi Computazionali della Fisica". Università di Milano (Dipartimento di Fisica).

Conoscenze necessarie
Si presuppone, nei partecipanti, una conoscenza matematica adeguata ai temi trattati, nonché una conoscenza di base dei fondamenti della programmazione procedurale.

Struttura della sezione dedicata alla programmazione ad oggetti
Il capitolo sulla parte di C++ è strutturato come segue:

- Un capitolo (sezione 5) dedicato ad introdurre i concetti fondamentali della programmazione ad oggetti in C++. Nella parte finale viene anche introdotto il concetto di "Design patter", una tematica più avanzata nell'ambito della programmazione ad oggetti e dello sviluppo di software professionale.

Bibliografia
Alla fine delle note è riportata una breve bibliografia riguardante la programmazione ad oggetti, che può essere utilizzata per approfondire gli argomenti trattati.

Capitolo 1

Elementi di finanza

1.1 Il contesto - Elementi di finanza

In questo primo capitolo definiremo alcuni elementi base della finanza (tasso di interesse, principio di non arbitraggio, attualizzazione dei flussi di cassa, definizione di "bond", concetto di opzione, ecc.). Una più esauriente e approfondita presentazione di questi argomenti può essere trovata in [1, 2].

1.1.1 Tasso di interesse

All'interno delle economie di mercato il denaro, se investito, riceve una remunerazione in base al cosiddetto tasso di interesse. Se ad esempio il tasso di interesse ad un anno è del 5%, ed investiamo una quantità di denaro pari a 100 EUR, fra un anno riceveremo: $100 \cdot (1 + 5/100) = 105$ EUR. Questo significa che il denaro nel nostro sistema economico tende a generare altro denaro. Il tasso di interesse misura questa capacità d'accrescimento. Ovviamente perché si verifichi questo fenomeno di moltiplicazione del capitale è necessario che il denaro venga investito (ovvero che diventi capitale). La teoria finanziaria non si occupa in ogni caso di esaminare attraverso quali meccanismi il capitale possa generare se stesso, nè di conseguenza è in grado di derivare quale sia il valore corretto del tasso di interesse. Semplicemente in finanza si assume, come dato sperimentale, che un capitale investito generi un interesse con un certo tasso. Sarà compito di una teoria economica stabilirne l'origine.
In generale quando si indica un tasso di interesse, va sempre specificato quale sia la frequenza di pagamento degli interessi. Si consideri ad esempio un capitale iniziale C_i su cui vengano pagati degli interessi r_{f_1} con una frequenza pari a f_1 volte l'anno per un totale di T anni. Il valore alla fine del periodo T, ipotizzando di re-investire gli interessi allo stesso tasso, sarà pari a:

$$C_f = C_i \left(1 + \frac{r_{f_1}}{f_1}\right)^{f_1 T} . \tag{1.1}$$

Se ora consideriamo una differente frequenza di pagamento, f_2, possiamo domandarci quale sia il tasso annuo r_{f_2} in grado di produrre lo stesso capitale finale C_f, ovvero quale sia il tasso r_{f_2} equivalente a r_{f_1}. La risposta è data dalla seguente espressione:

$$r_{f_2} = f_2 \left[\left(1 + \frac{r_{f_1}}{f1}\right)^{\frac{f_1}{f_2}} - 1\right] , \tag{1.2}$$

Da questa discussione si evince quanto sia importante specificare sempre la frequenza di pagamento degli interessi.
Dall'equazione (1.1), nel limite $f1 \to \infty$ (ovvero nel cosiddetto limite di

capitalizzazione continua), si ottiene la formula:

$$C_f = C_i \, e^{r_c T} \, , \tag{1.3}$$

Nel seguito supporremo sempre che il tasso di interesse faccia riferimento ad una capitalizzazione continua.

Ovviamente nel mondo finanziario reale qualunque investimento in grado di generare un interesse è caratterizzato da un certo rischio, più o meno elevato. Ad esempio concedere prestiti per l'acquisto di case (i cosiddetti mutui) ha un grado di rischio più alto di quello che si ha investendo in titoli di stato italiani (infatti è più facile che i mutuatari, per varie ragioni, non siano in grado di restituire il prestito ricevuto piuttosto che lo stato italiano fallisca). E d'altra parte, l'investimento in obbligazioni emesse dallo stato italiano è più rischioso dell'acquisto di titoli di stato americani. Generalmente, come del resto ovvio, si osserva che tanto più grande è il rischio di un investimento tanto maggiore sarà il tasso di interesse richiesto. D'altra parte possiamo attenderci che in un economia sviluppata, dove sia valido il principio di non arbitraggio (vedi paragrafo successivo), investimenti caratterizzati dalla stessa durata e dallo stesso rischio fruttino interessi uguali. Relativamente al rischio di un investimento, questo può essere definito come la probabilità che a scadenza l'investimento (e gli interessi generati) non vengano restituiti o vengano restituiti solo in parte.

Se indichiamo con $r_T(p)$ il tasso di interesse su un periodo di tempo pari a T e con p la probabilità che tale investimento vada completamente perso, allora si definisce **tasso di interesse privo di rischio**, il seguente limite:

$$\lim_{p \to 0} r_T(p) \tag{1.4}$$

Il tasso privo di rischio ("risk free rate"), pur essendo un'astrazione, è un concetto fondamentale della finanza ed è presente in tutti i modelli di *pricing* di strumenti finanziari sofisticati. In concreto possiamo immaginare che gli interessi corrisposti dai titoli di stato o pagati dalle banche sui depositi inter-bancari si avvicinino al tasso privo di rischio.
Un secondo elemento da considerare nella definizione data sopra, è l'intervallo di tempo T. Infatti in generale il tasso di interesse dipende dalla scadenza temporale considerata. E' quindi più corretto parlare di curva dei tassi *risk free*. Ad es. un investimento privo di rischio a lunga durata potrebbe essere caratterizzato da un rendimento annuo più basso o più alto di un investimento a breve durata.

1.1.2 Il principio del non arbitraggio

In questo paragrafo enunceremo una degli assiomi fondamentali della finanza. Vedremo come molti dei risultati che deriveremo nei prossimi capitoli

poggiano su questo assunto fondamentale.
Il principio in questione si enuncia nei seguenti termini:

Assioma del non arbitraggio: Nei mercati finanziari è impossibile realizzare un guadagno certo che abbia un rendimento superiore al tasso privo di rischio.

In gergo finanziario, questo principio è efficacemente sintetizzato affermando che: " There's no such thing as a free lunch in the market".

Si devono fare a questo proposito alcune considerazioni:

- questo principio vale in media, non si può infatti escludere che per brevi intervalli di tempo, possano comparire sul mercato delle opportunità di arbitraggio, ovvero *mis-pricing* di titoli finanziari che consentano di realizzare un guadagno certo superiore a quello dato dal tasso risk free. Naturalmente se il mercato è maturo ed efficiente, tali opportunità scompariranno in un tempo brevissimo. Facciamo un esempio concreto, considerando un titolo che venga quotato in due borse differenti, diciamo a Londra e a New York. Supponiamo che il cambio sterlina contro dollaro sia pari a 1,57 dollari per 1 sterlina e che le quotazione del titolo siano di una sterlina alla borsa di Londra e di 1.58 dollari a New York. In tale situazione si viene a realizzare un'opportunità di arbitraggio. Infatti acquistando ad esempio 100 titoli alla borsa di Londra al prezzo di 100 sterline e rivendendole immediatamente alla borsa di New York al prezzo di 158 dollari, ottengo un profitto istantaneo pari a 1 dollaro[1]. In tal caso avrei una violazione del principio di non arbitraggio, in quanto ho realizzato un guadagno certo in un tempo idealmente nullo (ottenendo quindi un rendimento infinito). Il mercato però tende a far scomparire queste opportunità in tempi rapidissimi. Ad esempio, nel caso considerato sopra, si verificheranno, per il titolo in oggetto, una serie di richieste d'acquisto alla borsa di Londra e contemporaneamente altrettante vendite a quella di New York. Per la legge della domanda e dell'offerta, questo si tradurrà in un aumento di prezzo del titolo sulla piazza di Londra e una diminuzione su quella di New York. In tal modo, in pochissimo tempo i due prezzi si riporteranno in equilibrio, impedendo ogni ulteriore profitto. Nel mercato esistono alcune persone la cui professione consiste proprio nello sfruttare piccoli disallineamenti di prezzo al fine di realizzare un guadagno. Questi *trader* prendono il nome di arbitraggisti. Paradossalmente, è proprio la loro esistenza che rende vero il principio di non arbitraggio.

[1]Supponiamo in questo caso che non vi siano costi di transazione. Questi sono generalmente un ostacolo allo sfruttamento di *mis-pricing* nel mercato.

- Esistono sul mercato diverse opportunità che consentono di realizzare un guadagno superiore al tasso privo di rischio, tipicamente gli investimenti azionari. Questi però sono sempre caratterizzati da un'incertezza nei risultati.

Come vedremo, il principio di non arbitraggio giocherà un ruolo chiave nell'argomentazione di Black & Scholes per l'*option pricing* e di fatto costituisce uno dei pilastri su cui si regge l'intera finanza quantitativa.

1.1.3 Attualizzazione del denaro

Da quanto discusso nel paragrafo 1.1.1, è chiaro che detenere 1000 EUR oggi non è la stessa cosa che avere 1000 EUR fra un anno. Un metodo molto semplice per rendere quantitative queste considerazioni consiste nell'introdurre il concetto di attualizzazione del denaro. In particolare proviamo a rispondere alla domanda: qual'è il valore attuale ("present value") ad oggi di un flusso di cassa futuro? Sia T l'intervallo di tempo che ci separa da un pagamento futuro pari a P_T, indichiamo con r il tasso di interesse privo di rischio, composto in maniera continua (cfr. eq. (1.3)), praticato dal mercato su un intervallo di tempo pari a T, allora il valore attuale di P_T è pari a:

$$P_0 = P_T \, e^{-r \cdot T} \ . \tag{1.5}$$

Dim.
In base al principio del non arbitraggio se il diritto a ricevere P_T fra un tempo T valesse più di P_0, diciamo $\bar{P}$, potrei vendere questo diritto ricevendo oggi $\bar{P}$ e impegnandomi a rimborsare P_T al tempo T, poi investirei il denaro $\bar{P}$ al tasso risk free. A scadenza, dopo T anni guadagnerei in maniera certa: $\bar{P}\, e^{r\,T} - P_T > 0$, senza aver impiegato alcun capitale iniziale[2] e questo sarebbe in contraddizione con il principio di non arbitraggio.
Analogamente se il valore del diritto a ricevere P_T fra un tempo T valesse meno di P_0, potrei prendere a prestito P_0 al tasso di mercato r impegnandomi a rimborsare P_T a scadenza. Una parte del denaro ricevuto oggi, $\bar{P}$, lo utilizzo per acquistare il diritto a ricevere P_T a scadenza, coprendo così il mio debito al tempo T. In tal modo ottengo oggi una quantità certa di denaro pari a: $P_0 - \bar{P} > 0$ senza nessun ulteriore obbligo e senza aver investito alcun capitale. Anche in questo caso avrei una violazione del principio di non arbitraggio. Resta quindi dimostrata la formula (1.5).

La formula (1.5) consente di poter dare un valore a flussi di denaro futuri. Nei fatti questo è uno dei risultati più fondamentali nel campo della finanza quantitativa, in quanto ogni strumento finanziario è caratterizzato sempre da flussi di cassa (più o meno certi) che verranno pagati in momenti

[2] Il rendimento dell'investimento sarebbe quindi infinito.

futuri. Generalmente gli strumenti finanziari si distinguono solo in base al meccanismo con cui vengono stabiliti i flussi futuri. Ad esempio in un titolo di stato (vedi paragrafo 1.1.4) che paga il 5% di interessi l'anno, i flussi futuri sono appunto le cedole annuali pari al 5% dell'investimento iniziale, più la restituzione del capitale a scadenza. Nel caso di un'azione, il suo valore è determinato dai tutti i dividendi che essa potrà corrispondere in futuro. E' quindi fondamentale poter disporre di una metodologia che ci consenta di dare un prezzo attuale a pagamenti che avverranno in futuro, fornendo in tal modo una misura omogenea. A questa richiesta risponde la formula (1.5).

1.1.4 Definizione di bond e sua valutazione

Utilizzando l'equazione (1.5) è relativamente semplice poter valutare gli strumenti a reddito fisso presenti sul mercato. A titolo di esempio consideriamo le obbligazioni a tasso fisso (bond). Questo è un titolo che viene sottoscritto dall'investitore ad un certo prezzo iniziale P_i e che paga con una certa frequenza f (tipicamente una o due volte l'anno) delle cedole pari a: $\frac{c}{f} N$, dove N è il nozionale acquistato e c è il tasso di interesse pagato dal bond. A scadenza, dopo T anni, il titolo rimborsa al sottoscrittore il suo valore nominale N. In altri termini il titolo garantisce all'acquirente il pagamento dei seguenti flussi monetari:

$$C_i = \frac{c}{f} N \quad \text{per } t = t_i = i/f \quad i = 1, ..., f\,T \tag{1.6}$$

$$N \text{ per } t = T \text{ (data di scadenza del bond)} , \tag{1.7}$$

dove T e t_i sono tempi espressi dimensionalmente in anni.
Se supponiamo di conoscere la curva dei tassi di interesse risk free, $r(t)$, e che non vi sia alcun rischio di mancato rimborso legato al bond, allora è molto semplice, utilizzando la formula (1.5), calcolare il valore P_i del bond:

$$P_i = N\, e^{-r(T)\,T} + \sum_{i=1}^{f\,T} C_i\, e^{-r(t_i)\,t_i} \ . \tag{1.8}$$

Come si vede ogni flusso di cassa futuro viene valorizzato ad oggi applicando il relativo fattore di sconto.

1.1.5 Curva dei tassi di interesse

Come si è evidenziato nel paragrafo precedente, il calcolo del prezzo di un titolo a tasso fisso presuppone la conoscenza della curva dei tassi. Questa può essere determinata sperimentalmente dai dati di mercato attraverso una procedura nota come "bootstrap". Questa metodologia prende come dati di partenza, una serie di prezzi di titoli obbligazionari quotati sul mercato (di

fatto quelli ritenuti più rappresentativi) e ricava la curva dei tassi che è in grado di produrre, tramite l'eq. (1.8), dei prezzi teorici, per ciascun titolo, identici ai rispettivi valori di mercato. Per ulteriori dettagli, rimandiamo al testo di Hull [1].

1.1.6 Titoli derivati e opzioni plain vanilla

1.1.6.a Titoli derivati

Def. Un titolo derivato è un contratto finanziario in cui l'oggetto della contrattazione è un altro strumento finanziario più fondamentale. Ne consegue che il valore di un titolo derivato dipende dal valore degli strumenti sottostanti su cui viene scritto il contratto.

Il sottostante di un contratto derivato è generalmente costituito da una variabile quotata sul mercato (ad es. può essere un'azione o un tasso di interesse o anche una materia prima quale ad esempio il petrolio) ma anche una grandezza d'altro tipo (purché ben definita) come ad es. la quantità di precipitazioni che si sono verificate in un certo periodo su una certa regione (i cosiddetti weather derivatives, che sono entrati recentemente anche sul mercato italiano). Non c'e' limite alla fantasia nello scrivere nuovi contratti derivati su sottostanti sempre più esotici.
I titoli derivati hanno conosciuto un'enorme crescita negli ultimi decenni e sono trattati sia nelle borse ufficiali, sia tra le grandi istituzioni finanziarie (i cosiddetti derivati *over the counter*, cioè sopra il banco). Inoltre diversi titoli obbligazionari, emessi da banche, contengono implicitamente contratti derivati.
Esistono diversi tipi di derivati. I più popolari sono i contratti futures, forward, IRS (Interest Rate Swap), cap&floor e le opzioni. In queste lezioni rivolgeremo la nostra attenzione unicamente alle opzioni.

1.1.6.b Definizione di opzione

Le opzioni furono introdotte per la prima volta nel 1973 e da allora hanno conosciuto una crescita di volumi inarrestabile.
Le più semplici sono le cosiddette opzioni plain vanilla. A tal riguardo, diamo la seguente definizione:

Def. Un'opzione dà diritto al possessore di acquistare o vendere, ad una data futura, un certo bene (il sottostante) ad un certo prezzo (detto prezzo di esercizio o *strike price*).

La prima cosa da osservare è il fatto che il proprietario di un'opzione ha acquistato un diritto e non un obbligo. Se ad esempio l'opzione dà diritto ad acquistare un certo asset finanziario ad un prezzo maggiore del suo valore di mercato, è ovvio che tale diritto non verrà esercitato. Questa è la

caratteristica principale che differenzia un'opzione da altri contratti derivati (ad es. forward o future).

Le opzioni si dividono in tre grandi famiglie per quanto riguarda le condizioni temporali in cui il diritto insito nell'opzione può essere esercitato:

(i) **Esercizio di tipo europeo**
per le opzioni di tipo europeo, il diritto può essere fatto valere solo in corrispondenza ad una prefissata ed unica data futura, detta data di maturità (in inglese *maturity date*).

(ii) **Esercizio di tipo americano**
in questo caso l'opzione può essere esercitata in qualunque momento compreso tra oggi e la data di maturità dell'opzione. È quindi chiaro che un'opzione americana, rispetto ad una europea con le stesse caratteristiche, offre un diritto maggiore e di conseguenza avrà un prezzo più alto.

(iii) **Esercizio di tipo bermudano** [3]
nelle opzioni bermudane, l'opzione può essere esercitata solo in certe date prestabilite. Se indichiamo con $\{T_i\}_{i=1,....,N}$ il set di date in cui il diritto è esercitabile, allora la data di maturità dell'opzione è definita come $T_{\text{mat.}} = Max\{T_i\}_{i=1,....,N}$. Un'opzione di tipo europeo può quindi essere vista come una particolare opzione bermudana in cui il set di date di esercizio si riduce alla sola data di maturità. È ovvio che a parità di caratteristiche un'opzione con esercizio bermudano avrà un valore superiore ad una di tipo europeo.
D'altra parte, anche un'opzione americana risulta un caso limite di una opzione bermudana, e precisamente quello in cui il set di date $\{T_i\}_{i=1,....,N}$ include tutti i giorni compresi tra oggi e la *maturity date*. In questo caso, sempre a parità di caratteristiche, un'opzione bermudana avrà un valore inferiore ad una americana, in quanto si ha una restrizione del diritto di esercizio da parte del proprietario.

Il bene finanziario oggetto di un contratto di opzione, si dice sottostante. Sempre riguardo alla nomenclatura utilizzata in finanza, un'opzione si dirà di tipo call se dà diritto ad acquistare il sottostante, mentre sarà di tipo put nel caso in cui dia diritto a venderlo.
A titolo esemplificativo, consideriamo un'opzione che dà diritto ad acquistare delle azioni ENEL fra 1 mese ad un prezzo di 4 EUR. Si tratta in questo caso di un'opzione europea, di tipo call, con *maturity* ad 1 mese ed uno *strike price* di 4 EUR.

[3] Per la precisione, le opzioni con esercizio bermudano non rientrano tra quelle *plain vanilla*, e andrebbero già considerate opzioni esotiche (vedi capitolo 1.4).

È ovvio che un'opzione verrà esercitata dal suo possessore solo se questo porta ad un profitto. Ciò significa che nel caso di un'opzione call il prezzo a scadenza del sottostante dovrà essere maggiore dello *strike price*, mentre per una put dovrà essere inferiore; diversamente il proprietario dell'opzione non eserciterà il suo diritto.
Se indichiamo con $S(0)$ il prezzo del sottostante al tempo $t = 0$, con T il tempo mancante alla scadenza dell'opzione e con E lo strike price, allora il profitto (detto *pay-off*) per un'opzione europea alla data di maturità (cioè al tempo T) sarà nel caso di una call:

$$\text{Pay-off}_{\text{call}} = Max\left[S(T) - E, 0\right] \ , \tag{1.9}$$

mentre per una put:

$$\text{Pay-off}_{\text{put}} = Max\left[E - S(T), 0\right] \ . \tag{1.10}$$

Il grafico dei due pay-off è riportato in figura 1.1 In altri termini alla da-

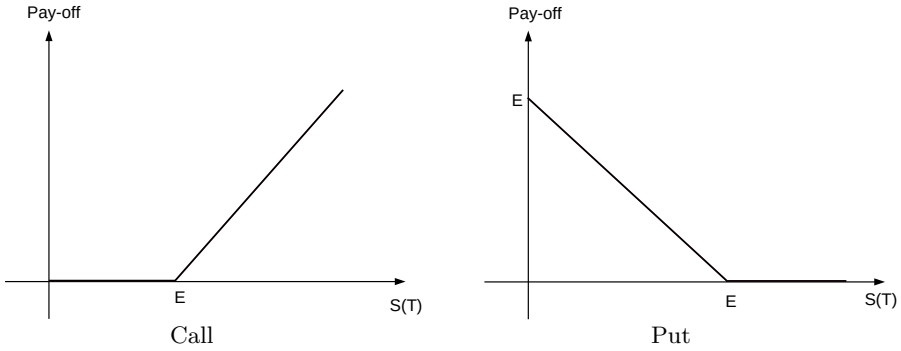

Figura 1.1: Pay-off *plain vanilla*(call e put).

ta di maturità, il valore dell'opzione diventa certo (ovvero deterministico) ed è stabilito dall'equazione riportata sopra. Viceversa per ogni istante di tempo precedente, il valore di questo diritto va valutato usando modelli probabilistici, perché nessuno è in grado di prevedere deterministicamente come evolverà il prezzo del sottostante. Il grande merito di Black e Scholes è stato quello di creare un modello scientifico razionale che consentisse di valutare correttamente il prezzo di un'opzione [13]. Da quanto detto è chiaro che il modello di Black & Scholes, così come qualunque altro modello che voglia stabilire il prezzo di un'opzione, si basa necessariamente su una teoria stocastica per descrivere il movimento dei prezzi del sottostante.

È interessante osservare che il pay-off di un'opzione call o put dipende solo dalla differenza tra il valore del sottostante ed il prezzo di esercizio E. Il sottoscrittore dell'opzione può realizzare questo profitto esercitando il proprio diritto, ovvero, nel caso di una call, acquistando l'azione al prezzo E e subito rivendendola al suo prezzo di mercato $S(T)$. Molto spesso l'opzione

viene regolata versando al suo proprietario direttamente il profitto $S(T) - E$ (nel caso di una call) evitando di consegnare fisicamente il sottostante. Le opzioni in cui il pay-off viene regolato in contanti, prendono il nome di *cash option*, mentre quelle dove viene effettivamente consegnato il sottostante, prendono il nome di *delivery option*. È abbastanza chiaro che, dal punto di vista del calcolo del prezzo, una *delivery option* è equivalente ad una *cash option*. Nel seguito, per semplicità, supporremo sempre di considerare opzioni che regolano il pay-off in modalità *cash*.

Infine dobbiamo osservare che un'opzione è un contratto e come tale coinvolge sempre due parti: chi acquista l'opzione e chi la vende (in gergo finanziario quest'ultimo viene indicato come colui che scrive l'opzione). Il possessore dell'opzione acquisisce un diritto e di conseguenza dovrà corrispondere, al tempo iniziale, un premio a chi ha venduto l'opzione. Alla data di maturità sarà chi ha scritto l'opzione a dover garantire il pagamento del pay-off in caso di esercizio da parte dell'acquirente. In gergo finanziario si afferma che chi vende l'opzione va corto e chi l'acquista va lungo. Di conseguenza nel caso di un'opzione di tipo call ci saranno due possibili posizioni: 1) call lunga (acquisto il diritto); 2) call corta (vendo il diritto, ovvero contraggo un obbligo, ricevendo in cambio un premio iniziale).
Si noti che a tutti gli effetti, un'opzione può essere vista come una sorta di assicurazione, in cui chi va lungo è l'assicurato e chi va corto è l'assicuratore. La principale differenza sta nel fatto che le opzioni possono essere utilizzate anche a fini puramente speculativi.

1.1.6.c Effetto leva delle opzioni

Come dovrebbe essere emerso in maniera abbastanza chiara dalla discussione condotta fino ad ora, le opzioni si caratterizzano per il loro elevato grado di incertezza. Il pay-off corrisposto da un'opzione a scadenza dipende dall'andamento del sottostante. Poiché il sottostante segue un'evoluzione stocastica (come illustreremo in maniera più precisa nel prossimo capitolo) è ovvio che anche il valore di un'opzione sarà caratterizzato da una forte componente probabilistica. Anzi per un'opzione la cosa è ancora più vera che per il sottostante. Consideriamo il seguente esempio: supponiamo di acquistare, in data odierna, un'azione al prezzo di 100 EUR. Supponiamo poi che a distanza di un anno l'azione venga quotata 120 EUR. In altri termini l'azione ha subito un incremento del 20%, ciò significa che investendo un capitale di 100 EUR abbiamo realizzato un guadagno di 20 EUR. Viceversa supponiamo di investire la medesima cifra di 100 EUR nell'acquisto di un'opzioni call su questa azione con *strike price* pari a 100 EUR e data di maturità fra un anno. Il prezzo della call sull'azione è di 10.002 EUR[4], quin-

[4]Il prezzo è ottenuto applicando la formula di Black & Scholes (vedi paragrafo 1.3.5), considerando un tasso $r = 3\%$ e una volatilità $\sigma = 21.5\%$. Abbiamo anche supposto che

di con 100 EUR possiamo comprare il diritto ad acquistare fra un anno circa 10 azioni al prezzo di 100 EUR. Perciò il guadagno netto che realizziamo dopo un anno (nel caso di un fortunato aumento del 20% nel prezzo dell'azione) è: $10\,(120-100)-100=100$ EUR, ovvero un guadagno del 100%! Questo comportamento è noto come effetto leva. Ovviamente poiché nel mercato, per il principio di non arbitraggio, non si hanno *free lunch*, ad una maggiore possibilità di guadagno corrisponde sempre un maggior rischio. In effetti nel caso di un calo nel prezzo dell'azione, la call option può andare *out of the money*[5] e l'investimento iniziale andare così completamente perso. Nell'esempio riportato sopra, se a distanza di un anno l'azione subisse un calo del 5%, l'investitore che ha acquistato l'azione perderebbe solo 5 EUR, mentre l'acquirente delle opzioni call perderebbe l'intero capitale.
In definitiva il mercato delle opzioni è notevolmente più rischioso e complesso di quello azionario. È quindi molto importante poter calcolare con precisione il valore di un'opzione. Mentre però nel caso di un bond la determinazione del prezzo si riduce a scontare flussi di cassa certi, nel caso di un'opzione si tratterà di scontare flussi di cassa incerti. È quindi necessario ricorrere al calcolo stocastico al fine di determinare probabilisticamente il valore del pay-off a scadenza, per poterlo poi scontare ad oggi. Come vedremo nel capitolo 1.3, per le opzioni *plain vanilla* è possibile ottenere delle formule chiuse per il prezzo del contratto.

l'azione non paghi dividendi durante l'anno.

[5] Un gergo finanziario per indicare che l'opzione non viene esercitata ed è perciò priva di valore

1.2 Processi stocastici per descrivere le azioni

Questo capitolo è dedicato alla discussione dei processi stocastici, con particolare riguardo a quelli markoviani e di Wiener; questi ultimi, come vedremo, svolgono un ruolo fondamentale in finanza.
Oltre a definire le diverse tipologie di processi stocastici e discutere le motivazioni che portano a modellizzare i movimenti azionari tramite processi di Wiener, presenteremo anche il famoso lemma di Ito che avrà un ruolo chiave nell'argomentazione di Black & Scholes per il *pricing* di un'opzione.

1.2.1 Introduzione

Una delle cose più sorprendenti che un fisico può incontrare leggendo un testo di finanza, è scoprire che il modello più largamente utilizzato per descrivere i movimenti casuali nei prezzi azionari è il ben noto modello browniano.
Da un punto di vista storico, il *random walk* fu introdotto da Einstein per giustificare il moto browniano cui è soggetto un granello di polline immerso in un liquido. Gli urti a cui il granello è soggetto da parte delle molecole d'acqua, porta ad un classico cammino casuale che è ben descritto dal modello *random walk* (vedi paragrafo 1.2.2). Facendo un parallelo tra il moto browniano della fisica e le oscillazioni nel valore di un'azione, possiamo immaginare che le informazioni che arrivino alla borsa svolgano lo stesso effetto degli urti delle molecole d'acqua, in altri termini ciascuna delle molte informazioni che pervadono il mercato finanziario ha l'effetto di portare ad una piccola variazione (al rialzo o al ribasso) nel prezzo dell'azione. L'azione combinata di tutte le informazioni fa si che il prezzo dell'azione si muova in accordo con un random walk. Naturalmente questa è solo una giustificazione qualitativa di carattere euristico, ed in effetti indagini più quantitative hanno messo in discussione questo modello (vedi paragrafo 1.2.6).
In questo capitolo introdurremo dapprima il modello *random walk*, considerandone il limite continuo, per poi passare alla discussione dei processi di Wiener e di Ito, entrambi sottoclassi dei più generali processi markoviani. Successivamente vedremo come i processi stocastici di Wiener possano essere utilizzati nell'ambito della finanza e con quali limiti. Concluderemo quindi con l'enunciazione e la dimostrazione del lemma di Ito.

1.2.2 Il modello random walk

Si consideri una variabile w che assume i valori:

$$\begin{array}{rll} & \Delta s & \text{con probabilità } p_{\text{left}} = p \\ - & \Delta s & \text{con probabilità } p_{\text{right}} = 1 - p \end{array} \tag{1.11}$$

Consideriamo ora una serie di salti, $\{w_i\}$, di ampiezza ΔS e durata Δt ed esaminiamo il valore assunto dalla variabile $x = \sum_{i=1}^{n} w_i$. Da un punto

di vista intuitivo la grandezza $x(t = n\,\Delta t)$ può essere riguardata come il cammino percorso in un tempo $t = n\,\Delta t$, da un ubriaco che si sposti aleatoriamente in accordo all'eq. (1.11).

È chiaro che anche x è una variabile aleatoria, e per il teorema del limite

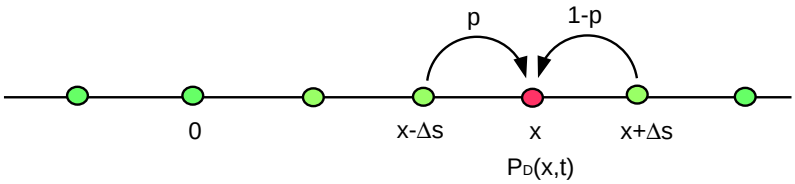

Figura 1.2: Random walk.

centrale sarà caratterizzata, nel limite $n \to \infty$, da una distribuzione gaussiana.

Se introduciamo la densità di probabilità nel discreto $P_D(x,t)$, definita come la probabilità di trovare il walker nel punto x al tempo t, allora possiamo scrivere per P_D la seguente equazione fondamentale [6]:

$$P_D(x, t + \Delta t) = p\, P_D(x - \Delta s, t) + (1 - p)\, P_D(x + \Delta s, t)\ . \tag{1.12}$$

con condizioni al contorno date da:

$$P_D(x, t = 0) = \delta_{x,0}\ ; \tag{1.13}$$

ovvero al tempo iniziale ($t = 0$) abbiamo la certezza di trovare il walker nell'origine ($x = 0$).

Riscrivendo p come: $p = \frac{1}{2} + k$, l'equazione sopra diventa:

$$\begin{aligned} P_D(x, t + \Delta t) - P_D(x, t) &= \frac{P_D(x + \Delta s, t) + P_D(x - \Delta s, t) - 2P_D(x, t)}{2} - \\ &- k\ [P_D(x + \Delta s, t) - P_D(x - \Delta s, t)]\ . \end{aligned} \tag{1.14}$$

Se ora consideriamo il limite del continuo, $\Delta t \to 0$ e $\Delta s \to 0$, utilizzando la seguente "ricetta":

$$\begin{aligned} n\,\Delta t &= t \quad \text{valore finito}\ , \\ \lim_{\Delta t \to 0} \frac{\Delta s^2}{\Delta t} &= b^2 \quad \text{valore finito}\ , \\ \lim_{\Delta t \to 0} 2\,k\,\frac{\Delta s}{\Delta t} &= a \quad \text{valore finito}\ , \\ P_D(x, t + \Delta t) &\approx P_D(x, t) + \frac{\partial P_D}{\partial t}\,\Delta t\ , \end{aligned} \tag{1.15}$$

[6] Vedi figura 1.2.

$$\begin{aligned} P_D(x \pm \Delta s, t) &\approx P_D(x,t) \pm \frac{\partial P_D}{\partial x}\,\Delta s + \frac{1}{2}\frac{\partial^2 P_D}{\partial x^2}\,\Delta s^2 \,, \\ P(x,t) &= \frac{P_D(x,t)}{\Delta s} \,, \end{aligned}$$

dove abbiamo indicato con $P(x,t)$ la densità di probabilità nel continuo. In effetti l'ultima equazione si giustifica osservando che:

$$1 = \sum_x P_D(x,t) = \sum_x \frac{P_D(x,t)}{\Delta s}\,\Delta s = \sum_x P(x,t)\,\Delta s = \int_{-\infty}^{+\infty} P(x,t)\,dx \,, \tag{1.16}$$

garantendo quindi la consueta normalizzazione della densità di probabilità nel limite del continuo.
Osserviamo infine che le condizioni (1.16) implicano: $k = \frac{1}{2}\,\frac{a}{b^2}\,\Delta S \sim \Delta S \to 0$; ovvero per avere un corretto limite nel continuo è necessario che la differenza tra la probabilità di salto a sinistra ($p_{\text{left}} = p$) e quella di salto a destra ($p_{\text{right}} = 1 - p$), svanisca nel limite in cui il passo reticolare ΔS vada a zero. Il modo in cui però $p_{\text{left}} - p_{\text{right}}$ tende a zero determina un diverso comportamente del modello nel continuo (ovvero diversi valori per i parametri a e b).
Con le trasformazioni indicate sopra, la *master equation* per la distribuzione di probabilità del walker si traduce, nel continuo, in un'equazione differenziale alle derivate parziali:

$$\frac{\partial P}{\partial t} = -a\,\frac{\partial P}{\partial x} + \frac{1}{2}\,b^2\,\frac{\partial^2 P}{\partial x^2} \,, \tag{1.17}$$

con la condizione al contorno:

$$P(x, t=0) = \delta(x) \quad \text{delta di Dirac.} \tag{1.18}$$

Se ora consideriamo il cambio di variabile:

$$\begin{cases} t' = t & , \\ x' = b^{-1}\,(x - a\,t) & , \end{cases}$$

l'equazione (1.17) si può riscrivere come:

$$\frac{\partial P}{\partial t'} = \frac{1}{2}\,\frac{\partial^2 P}{\partial x'^2} \,, \tag{1.19}$$

ovvero la classica equazione del calore.
La soluzione, con il vincolo (1.18), è la solita gaussiana:

$$P(x', t') = \frac{1}{\sqrt{2\pi t'}\,b}\,e^{-\frac{x'^2}{2t'}} \,, \tag{1.20}$$

con media nulla e varianza t'.
In termini delle variabili iniziali x e t:

$$P(x,t) = \frac{1}{\sqrt{2\pi t}\, b}\, e^{-\frac{(x-a\,t)^2}{2\,b^2\,t}} \ . \tag{1.21}$$

È quindi immediato dimostrare che:

$$\begin{aligned} \langle 1 \rangle &= \int_{-\infty}^{+\infty} P(x,t)\, dx = 1 \ , \\ \langle x \rangle &= \int_{-\infty}^{+\infty} x\, P(x,t)\, dx = a\, t \ , \end{aligned} \tag{1.22}$$

$$\langle (x - \langle x \rangle)^2 \rangle = \int_{-\infty}^{+\infty} (x - \langle x \rangle)^2 \ P(x,t)\, dx = b^2\, t \ , \tag{1.23}$$

in altri termini la distribuzione di probabilità $P(x,t)$ trasla con velocità uniforme $a\,t$ ed ha una deviazione standard pari a: $b\sqrt{t}$ (ovvero il ben noto risultato, secondo cui in un random walk la deviazione standard del cammino percorso, x, scala come la radice quadrata del tempo).
Se consideriamo un intervallo t molto breve, diciamo dt, possiamo scrivere:

$$dx = a\, dt + b dz \ , \tag{1.24}$$

dove dz è una variabile stocastica con varianza pari a dt.
Le due equazioni: (1.17) e (1.24), sono due diverse descrizioni dello stesso fenomeno. La prima rappresenta l'equazione differenziale alle derivate parziali per la distribuzione di probabilità della variabile $x(t)$, mentre la seconda costituisce l'equazione differenziale stocastica che descrive direttamente l'evoluzione di $x(t)$.

1.2.3 Processi di Wiener e di Ito

Il modello random walk visto nel paragrafo precedente, rappresenta la versione discretizzata di quello che comunemente viene chiamato processo di Wiener. A questo proposito diamo la seguente definizione:

Def. Si dice che una variabile stocastica x segue un processo di Wiener generalizzato se la variazione dx da essa subita in un intervallo di tempo infinitesimo dt, è data dalla seguente formula:

$$dx = a\, dt + b\, dz \, . \tag{1.25}$$

La costante a viene comunemente indicata come il drift del processo, mentre b prende il nome di volatilità. dz rappresenta la componente stocastica e

corrisponde ad un numero casuale estratto da una distribuzione normale con media nulla e varianza pari a dt. Possiamo dunque riscrivere dz come: $dz = w\sqrt{dt}$, dove w è una variabile aleatoria la cui densità di probabilità, $P(w)$, è una distribuzione gaussiana con varianza unitaria e media nulla:

$$P(w) = \frac{1}{\sqrt{2\pi}} e^{-\frac{w^2}{2}} dw \ . \tag{1.26}$$

Se F è una funzione di w, definiremo il valore di aspettazione di F come:

$$\langle F \rangle = \int_{-\infty}^{\infty} F(w)\, P(w)\, dw \ . \tag{1.27}$$

Considerando il cambio di variabile $dz = w\sqrt{dt}$, è immediato verificare che $\langle dz^2 \rangle = dt$.
Si osservi che nel caso un cui $b = 0$, l'equazione (1.25) si riduce ad una semplice equazione differenziale ordinaria, la cui soluzione è banalmente: $x(t) = x_0 + a\,t$. Il fatto che sia presente anche il termine dz cambia completamente la natura del problema, in quanto l'evoluzione di x non risulta più deterministica ma aleatoria, in altri termini, non si ha più a che fare con un'equazione differenziale ordinaria ma con un'equazione differenziale stocastica.

1.2.4 Lemma di Ito

Una ulteriore generalizzazione dei processi di Wiener sono i cosiddetti processi stocastici di Ito. Diamo la seguente definizione:

Def. Si dice che una variabile stocastica x segue un processo di Ito se la sua variazione dx in un intervallo di tempo infinitesimo dt è data dalla formula:

$$dx = a(x,t)\, dt + b(x,t)\, dz \tag{1.28}$$

In altri termini un processo generalizzato di Wiener è un caso particolare di un processo di Ito in cui le funzioni $a(x,t)$ e $b(x,t)$ sono costanti.

Relativamente a questi processi, un importante risultato è dato dal noto lemma di Ito:

Lemma di Ito Sia x una variabile stocastica che segua un processo di Ito con drift $a(x,t)$ e volatilità $b(x,t)$; sia $F(x,t)$ una funzione di x e del tempo t. Allora anche F segue un processo stocastico di Ito, con drift:

$$a\,\frac{\partial F}{\partial x} + \frac{\partial F}{\partial t} + \frac{1}{2}\, b^2\, \frac{\partial^2 F}{\partial x^2} \ , \tag{1.29}$$

e volatilità:

$$b\,\frac{\partial F}{\partial x} \ . \tag{1.30}$$

In questo paragrafo esporremo un'argomentazione euristica a sostegno del lemma di Ito.

Dim. La funzione F dipende dalle due variabili: x e t. A sua volta x segue un processo di Ito che dipende dal tempo e dalla variabile stocastica dz.
La variazione di F, a seguito di variazioni infinitesime in x e t, è:

$$dF = \frac{\partial F}{\partial x}dx + \frac{\partial F}{\partial t}dt + \frac{1}{2}\frac{\partial^2 F}{\partial x^2}dx^2 + \frac{1}{2}\frac{\partial^2 F}{\partial t}dt^2 + \frac{\partial^2 F}{\partial x \partial t}dx\,dt\ , \qquad (1.31)$$

dove abbiamo arrestato lo sviluppo di Taylor al secondo ordine.
Si noti che dx dipende dal tempo dt sia direttamente, sia indirettamente tramite dz. Ricordiamo infatti che dz è una variabile stocastica estratta da una distribuzione normale con media nulla e varianza pari a dt (ovvero $dz = w\sqrt{dt}$).
Tronchiamo ora lo sviluppo (1.31), trattenendo soli i termini deterministici di ordine dt e quelli stocastici di ordine $\sqrt{dt}$. In tal modo rimangono solo i fattori: dx, dt e dx^2. In effetti si potrebbe rimanere sorpresi dal fatto che dx^2 rientri tra i termini da conservare. Basta però osservare che: $dx^2 = b^2(dz)^2 + o(dt)$ e che $(dz)^2$ è una variabile aleatoria che ha media: $\langle (dz)^2 \rangle = dt$ e varianza:

$$\langle \left[(dz)^2 - \langle (dz)^2 \rangle \right]^2 \rangle \sim dt^2\ , \qquad (1.32)$$

pertanto la parte stocastica di dx^2, essendo di ordine dt, è trascurabile rispetto a $dz \sim \sqrt{dt}$ ma la sua parte deterministica (approssimabile con $\langle (dz)^2 \rangle$), essendo di ordine dt, non può essere scartata.
In definitiva:

$$dF = \left(a\,\frac{\partial F}{\partial x} + \frac{\partial F}{\partial t} + \frac{1}{2}\,b^2\,\frac{\partial^2 F}{\partial x^2} \right) dt + b\,\frac{\partial F}{\partial x}\,dz\ . \qquad (1.33)$$

c.v.d.

Questo risultato è particolarmente utile nel caso dello studio di opzioni, in quanto come vedremo fra breve, le azioni sono modellate tramite un processi di Wiener generalizzato e poiché le opzioni non sono altro che funzioni del proprio sottostante è chiaro che ad esse può essere applicato il lemma di Ito.

Come curiosità ci si potrebbe chiedere se sia possibile scrivere un'opzione che abbia come sottostante un'altra opzione. Questo non solo è concettualmente possibile ma viene anche realizzato nella pratica; sul mercato si trovano a volte opzioni che consentono alla data di maturità di acquistare o vendere opzioni su un certo sottostante (si parla in tali casi di opzioni composte). Perciò a partire da un sottostante reale (ad es. un'azione) è idealmente possibile costruire una intera gerarchia di opzioni che possiamo indicare con $\{O^{(i)}\}$, dove l'opzione i-esima ha come sottostante l'opzione

(i-1)-esima e $O^0 \equiv S$ rappresenta un'azione. La cosa interessante è che se ci limitiamo a considerare opzioni in cui il valore del pay-off dipende solo dal valore del sottostante alla data di maturity[7], il prezzo dell'opzione O^1 sarà funzione unicamente del prezzo del sottostante S e del tempo t. Perciò in base al lemma di Ito anche il valore di $O^{(1)}$ seguirà un processo di Ito, e per induzione lo stesso sarà vero per ogni $O^{(i)}$, $i > 1$. Esistono però molte opzioni in cui il pay-off dipende dalla storia del sottostante[8] e non solo dal suo valore finale (queste opzioni sono dette path dependent). In questi casi il prezzo dell'opzione, istante per istante, non è detto che segua un processo di Ito e perciò la gerarchia creata sopra potrebbe condurre a variabili stocastiche non appartenenti alla famiglia dei processi di Ito.

1.2.5 Processi di Markov

I processi di Wiener generalizzati ed i processi di Ito, sono a loro volta casi particolari di una più ampia classe di processi stocastici detti di Markov.
Un processo markoviano è caratterizzato dal fatto che la variazione dx della variabile stocastica x in un intervallo di tempo dt, dipende solamente dal valore assunto da x al tempo t e non dalla sua storia precedente. In altri termini il cammino seguito dalla variabile per giungere al suo valore attuale non influenza la sua evoluzione futura.
I processi stocastici che abbiamo esaminato nei paragrafi precedenti sono casi molto particolari di processi markoviani in cui la componente casuale segue una distribuzione gaussiana. Esempi di processi markoviani che non rientrano in quelli di Ito sono ad esempio i processi di Levy, che hanno assunto una certa importanza in finanza [4]. Come abbiamo visto nei paragrafi precedenti, i processi di Wiener generalizzati sono l'equivalente continuo di un *random walk* e sono caratterizzati da un distribuzione probabilistica gaussiana che evolve nel tempo con una varianza proporzionale a t. Questi processi vengono anche detti diffusivi in quanto la loro varianza evolve linearmente col tempo. Dal punto di vista discreto, un processo di Wiener è caratterizzato da una particella che compie salti casuali, ciascuno dei quali è relativamente piccolo (nel modello del paragrafo 1.2.2, assumevamo che ogni salto avesse ampiezza costante $\pm\Delta s$). Di contro si potrebbe esaminare un processo discreto, in cui a salti di piccola entità si inframmezzano salti maggiori (*flights*). In tale situazione, la particella esplorerà una porzione di spazio più grande (il processo così originato si dirà di tipo super-diffusivo), e il percorso risultante sarà caratterizzato da linee raggomitolate, molto simili a quelle generate da un *random walk*, disconnesse le une dalle altre da salti bruschi. I processi di Levy non sono altro che l'equivalente continuo di questi modelli discreti e sono caratterizzati da una varianza che tende a divergere.

[7] Ovvero opzioni con esercizio europeo non *path dependent*
[8] Ad esempio nel caso di opzioni asiatiche (vedi paragrafo 1.4.1).

1.2.6 Il processo log normale come modello per le azioni

Una delle assunzioni fondamentali che vengono fatte in finanza è che l'evoluzione nei prezzi di una azione, segua un processo di tipo markoviano. Ciò significa che ai fini del mercato, l'unica cosa che conta per il futuro di un un'azione è il suo prezzo corrente. In altri termini, tenendo presente che le variazioni nei prezzi azionari sono di tipo aleatorio, ipotizzare per tali movimenti un processo di Markov equivale ad assumere che la distribuzione probabilistica per il prezzo futuro di un'azione dipenda unicamente dal suo prezzo corrente.
Poiché le variazioni nei prezzi delle azioni sono determinate dalla informazioni che arrivano alla borsa, l'assunzione precedente è in stretta relazione con la cosiddetta ipotesi di efficienza debole del mercato. Questa afferma che tutte le informazioni disponibili sul mercato sono racchiuse completamente nel prezzo attuale delle azioni.
Una conseguenza immediata di quanto affermato sopra, è l'infondatezza della cosiddetta analisi tecnica. Questa è una pratica abbastanza diffusa in finanza, la quale pretende di fornire delle previsioni sugli andamenti azionari futuri, partendo dall'analisi dei grafici delle serie storiche dei prezzi azionari. I sostenitori dell'analisi tecnica, asseriscono di poter realizzare guadagni maggiori, grazie alle informazioni ricavate da tale pratica.

Comunque. ad avviso del sottoscritto un mercato efficiente, ovvero un mercato in cui sono eliminate le opportunità di arbitraggio, non implica automaticamente il fatto che tutti i titoli in esso trattati seguano necessariamente processi markoviani. Semplicemente i processi stocastici che caratterizzano i vari strumenti, sono tali da non consentire, tramite l'analisi delle serie storiche, la realizzazione di un guadagno certo maggiore del tasso *risk free*. Esistono ovviamente molti cammini stocastici non markoviani che godono di questa caratteristica. Vedremo in particolare nel paragrafo 4.1.3 come sia possibile definire strumenti di mercato che pur non violando l'ipotesi di assenza di arbitraggio, ugualmente non seguono un cammino markoviano (e dove quindi la storia dei prezzi è importante per conoscere l'evoluzione futura). In conclusione, la vera motivazione in base alla quale rifiutare la teoria dell'analisi tecnica, è il fatto che questa violerebbe clamorosamente l'ipotesi di non arbitraggio.

Generalmente non solo si suppone che le variazioni percentuali nei prezzi delle azioni[9] seguano un processo markoviano, ma addirittura che siano in accordo con un processo di Wiener generalizzato. Questa proposta fu formulata da Bachelier nel suo lavoro di tesi di dottorato ([6]). È sorprendente il fatto che Bachelier abbia introdotto e studiato il moto browniano quasi cinque anni prima di Einstein.
Ovviamente assumere per i ritorni nei rendimenti azionari una distribuzione

[9] Ovvero: $\frac{S(t+\Delta t)-S(t)}{S(t)} = \frac{\Delta S}{S}$

log-normale, è un'ipotesi molto forte. Al di là delle giustificazioni di carattere euristico, date nel paragrafo 1.2.1, è ovvio che tale assunzione vada sottoposta ad una attenta indagine empirica.
Relativamente all'assunzione di markovianità nella serie storica dei prezzi azionari, diverse sono gli studi che hanno esaminato l'autocorrelazione nei ritorni dei prezzi. I risultati ottenuti [4], mostrano che l'autocorrelazione nei ritorni azionari tende a sopravvivere solo per intervalli di tempo molto piccoli, generalmente dell'ordine di alcuni minuti. È quindi abbastanza giustificato descrivere le oscillazioni nei prezzi azionari, tramite processi markoviani.
Riguardo all'assunzione di log normalità nella distribuzione dei ritorni, la prima analisi approfondita fu condotta da Mandelbrot in un famoso articolo del 1963 [7]. Mandelbrot considerò la serie storica dei prezzi del cotone su un intervallo di tempo molto lungo, riscontrando una deviazione dal previsto comportamento log-normale in corrispondenza delle code della distribuzione. Le code di una distribuzione dei ritorni dei prezzi rappresentano gli eventi in qualche modo estremi, ovvero i grandi crolli o i forti guadagni. In accordo alla teoria standard che descrive le variazioni percentuali nei prezzi delle azioni tramite un processo browniano, si sarebbe portati a concludere che i grandi shock azionari (positivi o negativi) avverrebbero con una probabilità che tende a calare esponenzialmente. In effetti l'indagine di Mandelbrot dimostrò che le forti oscillazioni di borsa sono più probabili di quanto non prescriva la teoria ordinaria. In particolare Mandelbrot propose di descrivere la distribuzione dei ritorni tramite un processo stabile di Levy. Questi processi stocastici sono caratterizzati da delle distribuzioni di probabilità con code (eventi estremi) che presentano un andamento a legge di potenza (tipo $1/x^{\alpha}$) e che sono quindi più "grasse" delle corrispondenti code esponenziali dei processi di Wiener [8].
Successive analisi empiriche sulle serie finanziarie, hanno confermato l'inadeguatezza del modello proposto da Bachelier. Attualmente la ricerca si concentra nello studio di dinamiche stocastiche alternative per descrivere la distribuzione dei ritorni azionari [10]. Resta comunque il fatto che, al momento, in finanza il modello correntemente utilizzato rimane quello log-normale. Questo ha due importanti conseguenze pratiche: 1) da un lato si tende a sottostimare i rischi di possibili perdite; 2) dall'altro si commetterà un certo errore nel pricing di strumenti derivati.

In conclusione il modello standard utilizzato in finanza per descrivere i movimenti azionari può essere enunciato come segue:

[10] Alcuni articoli interessanti che costituiscono un buon esempio del tipo di ricerca oggi condotta sul tema dell'analisi delle serie storiche finanziarie e delle leggi di scala che le governano, sono riportati nelle referenze [9, 10, 11, 12].

Def. Sia $S(t)$ il prezzo di un'azione al tempo t. Le variazioni nel prezzo dell'azione sono descritte dal seguente processo di Ito:

$$\frac{dS}{S} = \mu\, dt + \sigma\, dz \tag{1.34}$$

In altri termini le variazioni percentuali dei prezzi azionari sono caratterizzate da un termine deterministico, proporzionale a μ, e da un termine aleatorio gaussiano.
La costante μ prende il nome di fattore di drift mentre la costante σ viene indicata in finanza con il termine volatilità (nel seguito utilizzeremo sempre per la volatilità il simbolo σ).
Dalla definizione data sopra, e ricordando il lemma di Ito, si deduce immediatamente che:

$$d\log S = \left(\mu - \frac{\sigma^2}{2}\right) dt + \sigma dz\ , \tag{1.35}$$

ovvero il logaritmo dei prezzi è caratterizzato da un processo di Wiener generalizzato; si dice anche che il prezzo dell'azione segue un processo log-normale.
Dall'equazione (1.35) si può ottenere immediatamente la soluzione in forma chiusa per il processo stocastico (1.34):

$$S(T) = S_0\, e^{\left(\mu - \frac{\sigma^2}{2}\right)T + \sigma\sqrt{T}\, w}\ , \tag{1.36}$$

dove S_0 è il prezzo dell'azione al tempo zero e w è una variabile stocastica estratta da una distribuzione gaussiana con media nulla e varianza unitaria. Si può facilmente dimostrare che la media di S, $\langle S(T)\rangle$ è data da $e^{\mu\, T}$. Questo consente di interpretare il termine di drift, μ, come il rendimento medio atteso di S su un periodo T.

1.2.7 Limiti del modello diffusivo log-normale nella descrizione dei movimenti azionari

Il modello diffusivo log-normale applicato alla descrizione dell'evoluzione dei prezzi azionari, presenta alcuni limiti e difetti. Da un punto di vista strettamente attinente all'analisi dei dati, ci si potrebbe domandare se effettivamente i ritorni logaritmici azionari (calcolati su un certo intervallo di tempo Δt) siano effettivamente distribuiti secondo una gaussiana. Le analisi empiriche hanno mostrato in modo evidente (si faccia riferimento ad esempio ad alcuni lavori di Mandelbrot sul prezzo del cotone) una deviazione della densità di probabilità reale dal modello gaussiano. In particolare la maggiore discrepanza riguarda la presenza delle cosiddette code grasse (-fat tail), con particolare enfasi sulla parte a sinistra della distribuzione di probabilità. In sostanza il modello gaussiano prevederebbe che le code della distribuzione (che corrispondono quindi a forti incrementi o decrementi nel prezzo dell'azione) decadano esponenzialmente; questo implica, secondo il modello gaussiano, che la probabilità che si verifichino crolli o crescite di eccezionale entità sia sostanzialmente nulla. Per fare un esempio, se il modello lognormale descrivesse effettivamente l'evoluzione nei prezzi delle azioni, la crisi del 1929, da un punto di vista statistico, non si sarebbe mai dovuta verificare. Nella realtà le cose vanno però in modo diverso, gli eventi estremi si verificano con una frequenza ben superiore a quella prevista dal semplice modello lognormale. Tutto questo ha portato ad introdurre modelli più sofisticati per descrivere l'evoluzione dei prezzi azionari. Tra questi vi è certamente il modello a salti, l'oggetto del prossimo paragrafo.

1.2.8 Il processo a salti: definizione

Il modello a salti prende ispirazione da un fatto empirico ben noto: nella realtà le azioni possono subire salti repentini e non hanno necessariamente un andamento continuo (cosa invece implicata dal processo lognormale che produce curve di evoluzione sempre continue).
L'idea del modello a salti è quella di aggiungere al classico termine diffusivo (responsabile dell'evoluzione continua dell'azione) anche un termine di salto (jump in inglese) che mima la possibilitá di avere un salto improvviso e discontinuo nel valore dell'azione.
L'equazione stocastica del modello è la seguente:

$$\frac{dS}{S} = \mu\, dt + \sigma\, dz + (J-1)\, dN \tag{1.37}$$

Il primo termine è il classico termine di drift, il secondo rappresenta il processo diffusivo lognormale ed infine l'ultimo termine rappresenta il processo di salto. Durante l'intervallo di tempo dt, il termine dN può assumere due valori soltanto: 0 (assenza di salto) con probabilità $1 - \lambda\, dt$, oppure 1 (presenza di un salto) con probabilità $\lambda\, dt$. λ è l'intensità del salto.

Il termine di drift contiene un parametro μ al posto del classico tasso risk free (nella valutazione di un derivato si considera sempre un mondo neutrale al rischio). Infatti il termine μ viene determinato in modo tale da assicurare la seguente condizione (che è la condizione di neutralità al rischio):

$$< \frac{dS}{S} >= r\,dt \tag{1.38}$$

che si traduce nella seguente relazione tra i parametri del modello:

$$\mu = r - \lambda\,(J-1) \tag{1.39}$$

1.2.9 Il processo a salti: formula di pricing esatta

Il modello espresso dall'equazione (1.37) può essere risolto esattamente con una formula analitica data dalla somma (infinita) di prezzi di call (o put) alla Black & Scholes (con i parametri opportunamente ridefiniti).

1.3 Analisi di Black Scholes per il pricing delle opzioni

1.3.1 Introduzione

Questo capitolo è dedicato alla formula di Black & Scholes (nel seguito indicata con l'acronimo BS). Questa formula è estremamente importante sia per ragioni storiche, sia pratiche. Fu la prima derivazione quantitativa di una formula per valutare il prezzo di un'opzione [13]. Da un punto di vista pratico la formula di BS è ancora oggi largamente utilizzata in finanza.
Il presente capitolo inizia esponendo in dettaglio l'argomentazione di Black & Scholes per il *pricing* di opzioni europee. Discuteremo poi il significato di valutazione neutrale verso il rischio, un concetto chiave nel *pricing* dei derivati.
Ricondurremo quindi l'equazione differenziale di Black & Scholes all'equazione del calore che si è soliti incontrare in fisica e presenteremo la sua soluzione, la famosa formula di BS, nel caso di un'opzione *plain vanilla*. Infine sottoporremo ad analisi critica, le principali assunzioni su cui si basa la derivazione di BS.

1.3.2 L'argomento di Black & Scholes

Nel 1973 compare sulla rivista *Journal of Political Economy* un articolo firmato da F. Black e M. Scholes [13]. Questo *paper*, che porterà i suoi autori al nobel, segna una svolta nel campo dell'*option pricing* e più in generale della finanza quantitativa. Con esso, per la prima volta, veniva derivata una formula analitica (la cosiddetta formula di Black & Scholes) per calcolare il valore di un'opzione *plain vanilla*. Dopo quasi trenta anni dalla pubblicazione di quell'articolo, la formula di Black & Scholes viene ancora utilizzata attivamente dagli operatori finanziari, nonostante i suoi evidenti limiti (che discuteremo successivamente in questo capitolo).
L'argomentazione di Black & Scholes per il calcolo del prezzo di un'opzione europea *plain vanilla*, si basa su una serie di presupposti. Enunceremo in modo esplicito tutte le assunzioni fatte da BS al fine di stabilire i limiti di validità dei loro risultati.
Le principali assunzioni su cui si basa l'argomentazione standard di BS sono:

(A) il sottostante segue un processo di Wiener generalizzato; inoltre la volatilità è un parametro costante;

(B) la curva dei tassi è costante

(C) non esistono costi di transazione per operare sul mercato;

(D) si suppone che le azioni possano essere acquistate e vendute anche in valori frazionari;

(E) l'azione sottostante non paga dividendi durante la vita dell'opzione;

(F) il mercato segue perfettamente l'ipotesi di non arbitraggio.

La derivazione dell'equazione differenziale di Black & Scholes parte considerando un'azione, il cui prezzo sia $S(t)$, e un derivato qualsiasi costruito su di essa, il cui valore indichiamo con F. Se ci limitiamo a considerare un derivato che non sia *path dependent*, allora certamente possiamo assumere che $F = F(S, t)$.
Se ora utilizziamo l'assunzione (A) e ricordiamo il risultato fornito dal lemma di Ito, possiamo scrivere le equazioni stocastiche che descrivono le variazioni di S e F:

$$\begin{cases} dS = \mu S dt + \sigma S dz & , \\ dF = \left(\mu\, S\, \frac{\partial F}{\partial S} + \frac{\partial F}{\partial t} + \frac{1}{2}\, S^2\, \sigma^2\, \frac{\partial^2 F}{\partial S^2}\right) dt + \sigma\, S\, \frac{\partial F}{\partial S}\, dz & , \end{cases}$$

dove il punto essenziale è dato dal fatto che il processo stocastico dz è lo stesso in entrambe le equazioni. Questo fatto ci permette di considerare una combinazione lineare di S e F in cui non si abbia dipendenza da dz, ovvero in cui l'evoluzione sia completamente deterministica. In particolare supponiamo di definire un portafoglio composto dal derivato meno un numero di azioni pari a Δ (sfruttiamo a questo proposito l'ipotesi (D)):

$$\Pi = F - \Delta S \tag{1.40}$$

La variazione nel prezzo di Π in un intervallo di tempo infinitesimo è:

$$d\Pi = \left(\mu S \frac{\partial F}{\partial S} + \frac{\partial F}{\partial t} + \frac{1}{2}\,\sigma^2\, S^2 \frac{\partial^2 F}{\partial S^2} - \Delta\,\mu\, S\right) dt + \left(\sigma\, S\, \frac{\partial F}{\partial S} - \Delta\,\sigma\, S\right) dz\ . \tag{1.41}$$

La variazione nel prezzo del portafoglio Π può essere resa puramente deterministica scegliendo per Δ il seguente valore:

$$\Delta = \frac{\partial F}{\partial S} \tag{1.42}$$

naturalmente il valore di Delta potrà essere ritenuto costante, e pari al valore indicato dall'eq. (1.42), solo per un intervallo infinitesimo dt, in quanto la derivata di F rispetto a S tende a cambiare con il tempo. Sarà quindi necessario ri-eseguire un continuo ribilanciamento al fine di mantenere il portafoglio privo della componente stocastica (si parla in questo caso di *hedging* della posizione)[11].
Poiché il portafoglio evolve in maniera deterministica, per il principio di non

[11] Da un punto di vista teorico il ribilanciamento va effettuato in maniera continua. Questo non comporta alcun problema nel ragionamento di BS, in quanto ipotizziamo (punto (C)) che non vi siano costi di transazione.

arbitraggio (ipotesi (E)) la variazione $d\Pi$ deve essere la stessa che si avrebbe nel caso di un investimento privo di rischio, ovvero: $d\Pi = \Pi\, r\, dt$.
Otteniamo quindi l'equazione differenziale alle derivate parziali di Black & Scholes:

$$\frac{\partial F}{\partial t} + rS\frac{\partial F}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 F}{\partial S^2} = rF\ . \tag{1.43}$$

Si osservi che questa equazione è valida per un generico derivato, il cui prezzo dipenda unicamente da S e t[12]. La differenza tra un'opzione e l'altra è rimandata alle condizioni al contorno che si impongono su F al fine di risolvere l'equazione (1.43). In particolare, come abbiamo visto nel capitolo precedente, quello che caratterizza principalmente un'opzione è la formula che assegna il pay-off finale sulla base del prezzo raggiunto dal sottostante. Pertanto la condizione al contorno a cui deve soddisfare F è data da:

$$F(S, t = T) = \text{Pay-off}(S) \tag{1.44}$$

Il sistema formato da (1.43), completata con la condizione (1.44), sono sufficienti a derivare un'unica soluzione. Nel paragrafo 1.3.5 vedremo come sia possibile risolvere esattamente l'equazione di Black & Scholes nel caso di un'opzione europea *plain vanilla* (put o call).
È interessante sottolineare l'estrema semplicità dell'argomentazione di Black & Scholes, che pur consistendo di pochissimi passaggi ha portato ad un risultato notevole, premiato con il Nobel. Di più, essa fornisce una ricetta precisa, utilizzata in tutto il mondo finanziario, per annullare il rischio connesso con un'opzione. Infatti abbiamo visto come sia possibile creare un portafoglio, basato su una combinazione del derivato e del suo sottostante, in cui la componente stocastica sia assente. È chiaro che questa proprietà del portafoglio, Π, viene mantenuta solo per un intervallo di tempo infinitesimo, in quanto $\Delta = \partial F/\partial S$ tende a cambiare col tempo. È perciò necessario effettuare un continuo ribilanciamento al fine di mantenere il portafoglio privo di rischio. Nella pratica, questa operazione non può essere effettuata in maniera continua, sia per motivi pratici sia perché esistono dei costi di transazione (trascurati nell'argomentazione di BS). Ciò fa si, che i portafogli nella realtà non siano mai completamente privi di rischio. Ci si potrebbe domandare quale sia il senso di avere un portafoglio non soggetto a componenti aleatorie. Il motivo è riconducibile al fatto che le opzioni hanno un contenuto di rischio elevatissimo (come abbiamo spiegato nel paragrafo 1.1.6.c), ciò significa che a parte gli speculatori, tutti gli altri operatori di borsa hanno necessità di ridurre il rischio a cui sono esposti[13]; questo viene fatto

[12] Perciò in questa trattazione non sono inclusi i derivati *path dependent*, in cui il valore del derivato dipende anche dalla storia del sottostante.

[13] In particolare esistono normative molto severe che richiedono alle istituzioni finanziarie di accantonare dei capitali tanto più grandi quanto maggiori sono i rischi di mercato a cui si espongono. In generale qualunque sala operativa di una banca, fissa limiti precisi per l'esposizione al rischio. Quest'ultimo viene calcolato in maniera quantitativa tramite una metodologia nota come Var: valore a rischio.

utilizzando la tecnica di *hedging* descritta sopra.

1.3.3 Il concetto di valutazione neutrale verso il rischio

L'equazione di BS (1.43), dipende da alcuni importanti parametri quali il tasso di interesse privo di rischio r e la volatilità σ. Avremo modo nei prossimi paragrafi di discutere questi aspetti. La cosa però più interessante non è tanto da quali fattori dipende l'equazione di BS ma da quali non dipende. In particolare si può rimanere sorpresi dal fatto che nella formula (1.43) non compaia il termine μ, ovvero il tasso di crescita atteso per l'azione sottostante S! Ciò significa che il prezzo del derivato (ad esempio di un'opzione call) non dipende dal tasso di crescita del bene sottostante! Questo fatto, apparentemente controintuitivo, porta a definire un concetto chiave nella valutazione dei derivati, ovvero quello di mondo neutrale al rischio. Esaminiamo prima di tutto il significa di avversione al rischio. Investire su una azione comporta l'esposizione ad un certo rischio. L'azione infatti può aumentare, così come perdere considerevolmente di valore. In generale il mercato richiederà che l'azione fornisca un rendimento mediamente più alto del tasso *risk free*, come compenso al rischio a cui si è esposti. Questo giustifica il fatto che nel modello di evoluzione dei prezzi azionari, presentato nel paragrafo 1.2.6, abbiamo considerato un fattore di crescita, μ, distinto dal tasso privo di rischio r.
L'importante risultato, rappresentato dall'equazione di Black & Scholes, ha come corollario il fatto che il prezzo, F, del derivato non dipende da μ. Perciò è possibile effettuare il pricing di F ipotizzando che il mercato sia neutrale al rischio (ovvero ponendo $\mu = r$). Ciò comporta un'enorme semplificazione, come cercheremo di spiegare meglio nel prossimo paragrafo.

1.3.4 Alberi binomiali ad uno stadio e valutazione neutrale verso il rischio

In questo paragrafo andremo ad esaminare una situazione semplificata rispetto a quella considerata nel paragrafo 1.3.2. Questo al fine di discutere in dettaglio le argomentazioni che ci permettono di assumere una condizione di neutralità al rischio, durante la valutazione del prezzo di un derivato.
Supponiamo che oggi il valore di un'azione sia S_0 e che fra un periodo di tempo T, l'azione possa assumere unicamente due valori: S_{up} con probabilità p_{up} e S_{down} con probabilità p_{down} (vedi figura 1.3). Questo modello è ovviamente una semplificazione estrema rispetto alla situazione reale, però ci consentirà di esaminare con maggiore dettaglio quanto affermato nel paragrafo precedente.
Il valore medio atteso per l'azione S al tempo T è dato da:

$$\langle S(T) \rangle = p_{up}\, S_{up} + p_{down}\, S_{down} \ , \tag{1.45}$$

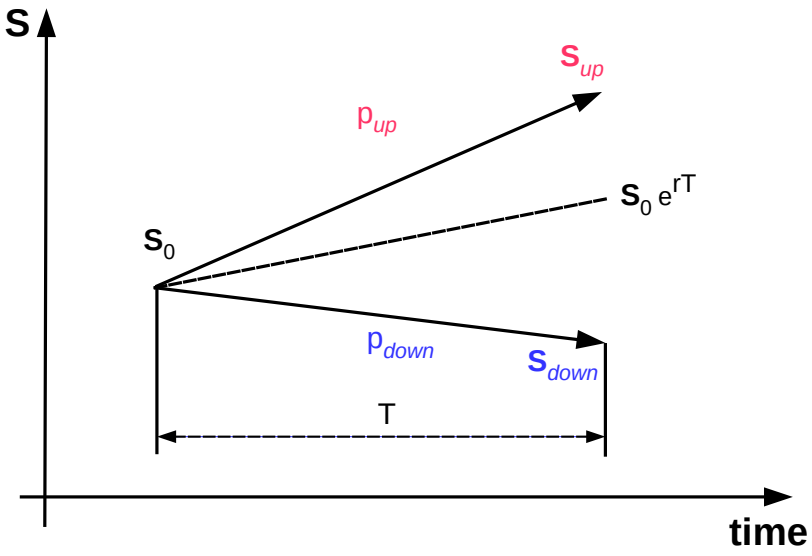

Figura 1.3: Albero binomiale ad uno stadio.

mentre la volatilità[14], nel prezzo futuro dell'azione, sarà:

$$\sigma = \sqrt{\langle (S(T) - \langle S(T) \rangle)^2 \rangle} = (S_{up} - S_{down}) \sqrt{p_{up}(1 - p_{up})} \, . \tag{1.46}$$

Poiché l'investimento in S è caratterizzato da una certa incertezza (data da σ), il mercato richiederà un rendimento medio atteso più alto di quello che si avrebbe in un caso privo di rischio:

$$\begin{aligned} &\langle S(T) \rangle = S_0 \, e^{\mu(\sigma) T} \\ &\mu(\sigma) > r \end{aligned} \tag{1.47}$$

Si dice in questi casi che il mercato è avverso al rischio e che di conseguenza richieda un premio per assumerne uno. Possiamo aspettarci naturalmente che: $\lim_{\sigma \to 0} \mu(\sigma) = r$. Si noti che l'interpretazione di μ come rendimento atteso della media della distribuzione, è esattamente l'analogo di quanto ottenuto nel paragrafo 1.2.6 .
Ovviamente se il mercato fosse completamente neutrale al rischio, ovvero se non si richiedesse alcun premio per assumere posizioni rischiose, si avrebbe $\mu = r$. Nella realtà, il mercato è avverso al rischio e quindi $\mu > r$, ovvero $S_{down} < S_0 \, e^{r \, T} < S_{up}$ (in figura 1.3, questa considerazione si traduce nel fatto che il segmento terminante nel punto $S_0 \, e^{r \, T}$, debba necessariamente rimanere confinato tra i due possibili scenari evolutivi S_{down} e S_{up}).

[14] Ovvero la deviazione standard

Consideriamo ora un derivato scritto sopra l'azione S. Il suo generico pay-off sarà definito come:

$$\begin{aligned} F(t=T) &= F_{up} \text{ se } S(T) = S_{up} \\ F(t=T) &= F_{down} \text{ se } S(T) = S_{down} \end{aligned} \tag{1.48}$$

Anche in questo caso possiamo costruire un portafoglio perfettamente privo di rischio, considerando una opportuna combinazione lineare del sottostante e del derivato:

$$\Pi = F - \Delta S \tag{1.49}$$

dove Δ è una costante scelta in modo da eliminare la componente stocastica dal portafoglio Π. In altri termini:

$$\Pi_{up} = F_{up} - \Delta S_{up} = \Pi_{down} = F_{down} - \Delta S_{down} \ , \tag{1.50}$$

da cui:

$$\Delta = \frac{F_{up} - F_{down}}{S_{up} - S_{down}} \ . \tag{1.51}$$

L'equazione (1.51) è esattamente l'analogo della condizione (1.42), dove al posto delle derivate parziali abbiamo delle differenze finite.
Se ora imponiamo che la variazione nel valore del portafoglio sia pari a quella di un investimento privo di rischio: $\Pi(T) - \Pi(0) = \Pi(0)e^{rT} - \Pi(0)$, otteniamo l'analogo dell'equazione di Black & Scholes:

$$F_{up} - \Delta S_{up} = e^{rT} \left(F_0 - \Delta S_0 \right) \ , \tag{1.52}$$

dove l'unica incognita è il valore del derivato F_0 ad oggi.
Si noti come, anche in questo caso, nell'equazione non compaiano le probabilità p_{up} e p_{down}. Ciò significa che il prezzo F_0 del derivato, non dipende dal tasso di crescita del valore atteso dell'azione, μ. Questo è esattamente lo stesso risultato che abbiamo incontrato nei paragrafi 1.3.2 e 1.3.3.
L'equazione (1.52) unita alle condizioni al contorno (1.48) può essere facilmente risolta, ottenendo:

$$F_0 = e^{-rT} \left[F_{up} p + F_{down}(1-p) \right] \ , \tag{1.53}$$

dove:

$$p = \frac{S_0 e^{rT} - S_{down}}{S_{up} - S_{down}} \ . \tag{1.54}$$

Si osservi che la valutazione di F_0 risulta totalmente indipendente dalla probabilità di rialzo effettiva, p_{up}.
Si può dare una semplice interpretazione della variabile p, invertendo l'equazione (1.45), ovvero esprimendo la probabilità p_{up} in termini di μ:

$$p_{up} = \frac{S_0 e^{\mu T} - S_{down}}{S_{up} - S_{down}} \ , \tag{1.55}$$

come si può vedere la variabile p coincide con la probabilità di rialzo dell'azione S, nel caso in cui il mercato fosse neutrale al rischio.
Se ora vogliamo trovare qual'è il rendimento atteso sul derivato F, basterà osservare che:

$$F_0\, e^{\gamma T} = \langle F(T) \rangle = F_{up} p_{up} + F_{down}(1 - p_{up}) \;, \tag{1.56}$$

sostituendo nell'equazione (1.56) la (1.53), otteniamo la relazione che lega γ a μ e r:

$$e^{\gamma T} = e^{rT} \frac{F_{up} p_{up} + F_{down}(1 - p_{up})}{F_{up} p + F_{down}(1 - p)} \;, \tag{1.57}$$

Come si vede il rendimento medio atteso sul derivato γ, dipende dal tasso risk free r e da μ (tramite p_{up}). Osserviamo che nel caso in cui il mercato sia neutrale al rischio ($\mu = r$) allora $\gamma = r$. È anche interessante esaminare il valore di γ in due semplici situazioni:

- **Call option**
  Supponiamo di considerare una call option, in cui lo strike price E sia compreso tra S_{down} e S_{up}. In tal caso: $F_{down} = 0$ e $F_{up} = S_{up} - E$. Dall'equazione (1.57) segue che:

  $$e^{\gamma_{call} T} = e^{rT} \frac{S_0 e^{\mu T} - S_{down}}{S_0 e^{rT} - S_{down}} \;. \tag{1.58}$$

  Poiché in un mondo avverso al rischio $\mu > r$, segue che $\gamma_{call} > r$.

- **Put option**
  Supponiamo di considerare una put option, in cui lo strike price E sia compreso tra S_{down} e S_{up}. In tal caso: $F_{up} = 0$ e $F_{down} = E - S_{down}$. Dall'equazione (1.57) segue che:

  $$e^{\gamma_{put} T} = e^{rT} \frac{S_{up} - S_0 e^{\mu T}}{S_{up} - S_0 e^{rT}} \;. \tag{1.59}$$

  Poiché in un mondo avverso al rischio $\mu > r$, segue che $\gamma_{put} < r$.

Il risultato ottenuto per γ nel caso di una *put option* può sembrare curioso, in quanto il rendimento per questo tipo di strumento, in un mondo avverso al rischio, è inferiore al tasso *risk free*. In realtà la cosa è perfettamente coerente, in quanto la put svolge all'interno del portafoglio Π, il ruolo di un'assicurazione contro i crolli azionari. Infatti se richiediamo all'azione S un rendimento mediamente più alto di r, per essere ripagati dell'incertezza a cui ci esponiamo, è abbastanza ovvio che un'assicurazione (put) che ci tuteli dai movimenti sfavorevoli, S_{down}, deve fornire un rendimento inferiore a r. In ogni caso però, il prezzo del derivato non dipenderà da μ. Se μ è più alto (ovvero se p_{up} è maggiore) la put avrà una minore probabilità di esercizio,

ovvero genererà un pay-off atteso minore, ma questo verrà attualizzato con un fattore di sconto più basso, non alterando quindi la stima di F_0.
Riassumendo, assumere un'avversione al rischio da parte degli investitori porta ad avere per l'azione un rendimento medio più alto del tasso risk free e per il derivato un rendimento che può essere più alto o più basso a seconda di come esso sia definito. La cosa però fondamentale e veramente importante, è che la stima del valore del derivato oggi non dipende da questi parametri. In altri termini ai fini del calcolo del prezzo del derivato, possiamo assumere che il mondo sia perfettamente neutrale al rischio (ovvero $\mu = r$) e valutare quindi il pay-off atteso (tramite il calcolo del valore di aspettazione su tutti i possibili scenari, ciascuno pesato con una probabilità neutrale al rischio) attualizzandone il valore ad oggi tramite il tasso risk free. Questa "ricetta", che trova la sua giustificazione teorica nelle formule (1.43) e (1.52), consente da un lato di semplificare notevolmente il problema del *pricing* di un'opzione e dall'altro permette di prescindere dalla valutazione di μ. Infatti mentre la stima del tasso *risk free* è relativamente agevole, valutare il valore di μ è tutt'altro che semplice.

1.3.5 Soluzione dell'equazione di Black & Scholes

Ritorniamo all'equazione di Black & Scholes (1.43). Come abbiamo avuto modo di spiegare nel paragrafo 1.3.2, questa equazione è valida per un generico derivato scritto sopra l'azione S. Il punto in cui entrano le caratteristiche dei diversi derivati, sono le condizioni al contorno (vedi eq. (1.44)).
L'equazione differenziale alle derivate parziali (1.43), completata con la condizione (1.44), può essere utilmente riscritta introducendo i seguenti cambi di variabile:

$$S = C\, e^x \,, \tag{1.60}$$

$$t = T - \frac{\tau}{1/2\,\sigma^2} \,, \tag{1.61}$$

$$F(S,t) = C\, v(x,\tau) \,, \tag{1.62}$$

dove per ora C è una costante generica.
Otteniamo così la seguente equazione differenziale:

$$\frac{\partial v}{\partial \tau} = \frac{\partial^2 v}{\partial x^2} + (k-1)\,\frac{\partial v}{\partial x} - k\, v \,. \tag{1.63}$$

con la condizione al contorno

$$v(x,0) = \mathcal{P}(x) \,. \tag{1.64}$$

La costante k è un parametro adimensionale dato da: $k = r/\frac{1}{2}\sigma^2$.
L'equazione (1.63) è molto simile all'equazione del calore della fisica, a parte

la presenza di termini in $\partial v/\partial x$ e in v. Questi possono essere facilmente eliminati introducendo una funzione $u(x,\tau)$ definita come:

$$u(x,\tau) = e^{-\alpha x - \beta\tau}\, v(x,\tau)\ . \tag{1.65}$$

ottenendo:

$$\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial u}{\partial x}\left[(k-1) + 2\alpha\right] + u\left[\alpha\,(k-1) + \alpha^2 - \beta - k\right]\ . \tag{1.66}$$

Basterà ora porre:

$$\alpha = -\frac{k-1}{2}\ ,$$
$$\beta = \alpha(k-1) + \alpha^2 - k = -\frac{(k+1)^2}{4}\ ,$$

per ottenere l'usuale equazione del calore:

$$\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2}\ . \tag{1.67}$$

L'equazione differenziale alle derivate parziali che definisce il pricing di un generico derivato di stile europeo è quindi la ben nota equazione del calore! Un risultato che mostra, ancora una volta, le similitudini che legano i modelli utilizzati in finanza con quelli della fisica.
È interessante ora considera un semplice caso, quello delle opzioni call europee *plain vanilla*. In questo caso la condizione al contorno data da:

$$F_{\text{call option}}(S, t=T) = Max\left[S(T) - E, 0\right]\ . \tag{1.68}$$

Se nel cambio di variabile (1.62) poniamo:

$$C = E\ , \tag{1.69}$$

la condizione (1.68) diventa:

$$u(x,0) \stackrel{def.}{=} u_0(x) = Max\left(e^{\frac{k+1}{2}x} - e^{\frac{k-1}{2}x}, 0\right) = \begin{cases} e^{\frac{k+1}{2}x} - e^{\frac{k-1}{2}x} & \text{se } x > 0 \\ 0 & \text{se } x \le 0 \end{cases}$$

La soluzione dell'equazione (1.67), con le condizioni al contorno date sopra, è:

$$u(x,\tau) = \frac{1}{2\sqrt{\pi\tau}} \int_{-\infty}^{\infty} u_0(s) e^{-\frac{(x-s)^2}{4\,\tau}} ds\ . \tag{1.70}$$

Infatti nel limite $\tau \to 0$ si ha:

$$\lim_{\tau\to 0} u(x,\tau) = \int_{-\infty}^{\infty} u_0(s)\delta(x-s)ds = u_0(x)\ , \tag{1.71}$$

La formula (1.70) può essere riscritta nella seguente maniera:

$$u(x,\tau) = I(x,\tau,k+1) - I(x,\tau,k-1)\ . \tag{1.72}$$

dove:

$$I(x,\tau,k\pm 1) = \frac{1}{2\sqrt{\pi\tau}} \int_0^\infty e^{\frac{k\pm 1}{2}s}\, e^{-\frac{(x-s)^2}{4\,\tau}} ds\ . \tag{1.73}$$

L'integrale I, può essere riscritto considerando un cambio di variabile:

$$s = a\,y + b\ . \tag{1.74}$$

dove a e b vanno determinare richiedendo che:

$$\frac{k+1}{2}s - \frac{(x-s)^2}{4\,\tau} = -\frac{1}{2}y^2 + f(x,\tau)\ . \tag{1.75}$$

da questa condizione deriva immediatamente che:

$$\begin{aligned} a &= \sqrt{2\,\tau}\ , \\ b &= 2\,\tau\left(\frac{k+1}{2} + \frac{x}{2\,\tau}\right)\ , \\ f(x,\tau) &= \frac{1}{4}\,\tau(k+1)^2 + \frac{1}{2}x(k+1)\ , \end{aligned} \tag{1.76}$$

e quindi:

$$I(x,\tau,k+1)) = e^{\frac{1}{4}\,\tau(k+1)^2 + \frac{1}{2}x(k+1)} N(d_1)\ , \tag{1.77}$$

dove:

$$N(d_1) = \frac{1}{\sqrt{2\pi}} \int_{-d_1}^{+\infty} e^{-\frac{y^2}{2}} dy\ , \tag{1.78}$$

e

$$d_1 = \frac{b}{a} = \sqrt{2\tau}\left[\frac{1}{2}(k+1) + \frac{x}{2\tau}\right]\ , \tag{1.79}$$

Se ora ri-sostituiamo a x, τ e u, le originarie variabili S, t e F_{call}, otteniamo:

$$F_{\text{call option}} = e^{-rT}\left[SN(d_1)e^{rT} - EN(d_2)\right]\ . \tag{1.80}$$

con:

$$d1 = \frac{\log\frac{S}{E} + \left(r + \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}}\ , \tag{1.81}$$

$$d2 = \frac{\log\frac{S}{E} + \left(r - \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}} = d_1 - \sigma\sqrt{T}\ . \tag{1.82}$$

La formula (1.80) può essere interpretata nel seguente modo: $N(d_2)$ è la probabilità in un mondo neutrale verso il rischio, che l'opzione venga esercitata, cosicché $E\,N(d_2)$ rappresenta il valore atteso per il prezzo di esercizio.

Viceversa, $SN(d_1)e^{rT}$ è il valore atteso per una variabile che risulti uguale a $S(T)$ se $S(T) > E$ ed è invece nulla in caso contrario.

Analogamente si può dimostrare che per un'opzione put il valore è dato dalla seguente formula:

$$F_{\text{put option}} = e^{-rT}\left[EN(-d_2) - SN(-d_1)e^{rT}\right] . \tag{1.83}$$

Il termine $N(d)$ che compare nelle formule di valutazione di Black & Scholes, è la ben nota funzione di distribuzione normale cumulata. Questa può essere valutata tramite funzioni di libreria oppure ricorrendo a formule approssimate. Riportiamo di seguito un schema per la sua valutazione, esatta fino alla sesta cifra decimale:

$$N(d) = \begin{cases} 1 - N'(d) * (a_1\,k + a_2\,k^2 + a_3\,k^3 + a_4\,k^4 + a_5\,k^5) & \text{quando } d \geq 0 \\ 1 - N(-d) & \text{quando } d < 0, \end{cases}$$

dove:

$$\begin{aligned} N'(d) &= \frac{1}{\sqrt{2\pi}}\, e^{-d^2/2} , \\ k &= \frac{1}{1 + \gamma * d} , \\ \gamma &= 0.2316419 , \\ a_1 &= 0.319381530 , \\ a_2 &= -0.356563782 , \\ a_3 &= 1.781477937 , \\ a_4 &= -1.821255978 , \\ a_5 &= 1.330274429 . \end{aligned}$$

1.3.6 Critica all'approccio di Black & Scholes

Volendo sottoporre ad analisi critica, il modello proposto da BS per il *pricing* di un'opzione, dobbiamo ritornare alle cinque assunzioni che ne stanno alla base (vedi paragrafo 1.3.2).
Di queste, alcune possono essere rese meno stringenti, in particolare è possibile derivare ugualmente delle formule di pricing esatte, nel caso in cui si abbia un tasso risk free che dipenda dal tempo in maniera deterministica. Ugualmente nel caso in cui $\sigma = \sigma(t)$[15]. Anche le assunzioni C, D e E non pongono problemi seri. Infine l'ipotesi F è un principio che difficilmente può essere messo in discussione. L'unica vera ipotesi che risulta determinante è l'assunzione di log-normalità per le oscillazioni del sottostante. A questo

[15] Anche in questo caso si deve però supporre che la dipendenza della volatilità dal tempo sia di natura deterministica. Se viceversa si introduce una volatilità stocastica, si perde la trattabilità analitica.

riguardo, abbiamo già evidenziato come le analisi empiriche mettano in discussione questo modello stocastico per le azioni.
Non stupisce quindi che le formule di BS non siano perfettamente in linea con il mercato reale. La cosa è particolarmente evidente nel calcolo della volatilità implicita, come ora spiegheremo.
Uno dei pregi maggiori del modello di BS è il fatto che questo dipenda da un unico parametro non direttamente osservabile: la volatilità. Quest'ultima può certamente essere calcolata utilizzando i dati storici dei prezzi del sottostante, ma rimane comunque una grandezza non direttamente quotata dal mercato (a differenza del tasso di interesse). Di fatto la formula di BS viene utilizzata dal mercato per ricavare, partendo dalle quotazioni delle opzioni trattate regolarmente, il valore di volatilità che dovrebbe avere il sottostante per produrre il prezzo osservato. In altri termini si inverte l'equazione di BS per ricavare σ. Questa stima prende il nome di volatilità implicita.
Se il modello di BS fosse corretto, ci si dovrebbe aspettare un valore di *sigma* implicito indipendente ad esempio dallo *strike price* delle opzioni considerate[16], questo perché la volatilità è una proprietà statistica del sottostante e non dell'opzione. In realtà, la volatilità implicita dipende da E, con un caratteristico andamento a *smile*. In pratica la volatilità sembra più alta se si considerano opzioni molto *in the money*[17] o *out of the money*[18] e minore per le opzioni *at the money*.
Per giustificare questo effetto di *smile*, è necessario considerare dinamiche alternative per il sottostante. A questo riguardo diverse sono le proposte avanzate [14].

[16] Ovviamente prenderemo in considerazione diverse opzioni con uguale *maturity*, scritte sul medesimo sottostante e che però abbiano differente *strike price*

[17] Nel caso di un'opzione di tipo call, si usa l'espressione *in the money* quando il prezzo ad oggi del sottostante è maggiore dello *strike price*. L'opposto nel caso di una put.

[18] Nel caso di un'opzione di tipo *call*, si dice che l'opzione e' *out of the money* quando il valore attuale del sottostante è minore dello *strike price*. Viceversa nel caso di una put.

1.4 Opzioni esotiche

Fino ad ora abbiamo considerato le cosiddette opzioni plain vanilla, ovvero opzioni standard. In generale il mercato nel corso degli anni ha sviluppato opzioni con caratteristiche più complesse rispetto alle semplici plain vanilla esaminate nel paragrafo 1.1.6. Queste opzioni più sofisticate prendono il nome di opzioni esotiche. Esistono una miriade di opzioni esotiche e, di fatto, una classificazione esaustiva è difficilmente realizzabile, in quanto ogni giorno ne vengono inventate di nuove. È comunque possibile illustrare le principali caratteristiche di quelle più diffuse.

1.4.1 Opzioni asiatiche

In queste opzioni, che hanno sempre esercizio di tipo europeo, il pay-off è definito confrontando lo strike price, non con il prezzo del sottostante a scadenza, ma con la media dei prezzi durante la vita dell'opzione. Le date di rilevazione da considerare al fine del calcolo della media possono corrispondere, per esempio, ad una frequenza mensile, piuttosto che ad un insieme di date non uniformemente distanziate.
Le opzioni asiatiche sono il tipico esempio di opzioni *path dependent*, in cui cioè il pay-off finale dipende non solo dal valore finale del sottostante ma anche dalla sua storia.

1.4.2 Opzioni digitali

Nelle opzioni di tipo *digital* il pay-off può assumere solo valori discreti. Un tipico esempio è l'opzione call (put) *cash or nothing* in cui il pay-off è pari a una quantità prefissata K se alla data di *maturity* il valore del sottostante è maggiore (minore) dello strike price, E, oppure a zero diversamente (vedi fig. 1.4).
Un altro esempio è l'opzione call (put) *asset or nothing*, in cui il pay-off è pari al valore del sottostante se alla data di maturity quest'ultimo è maggiore (minore) dello strike price, E, oppure a zero diversamente.
È chiaro che una call (put) plain vanilla con *strike*, E, è la somma di una call (put) *asset or nothing* meno una call (put) *cash or nothing* con K uguale allo *strike price* E.
Ovviamente gli esempi dati sopra sono molto semplici e per questi sono disponibili delle formule chiuse. In generale comunque è possibile trovare sul mercato opzioni digitali più complesse.

1.4.3 Opzioni con barriera

Queste opzioni sono un altro esempio di opzioni *path dependent*. Nelle opzioni con barriera il pay-off finale è condizionato dal fatto che il prezzo dell'attività sottostante abbia raggiunto o meno un certo livello (barriera).

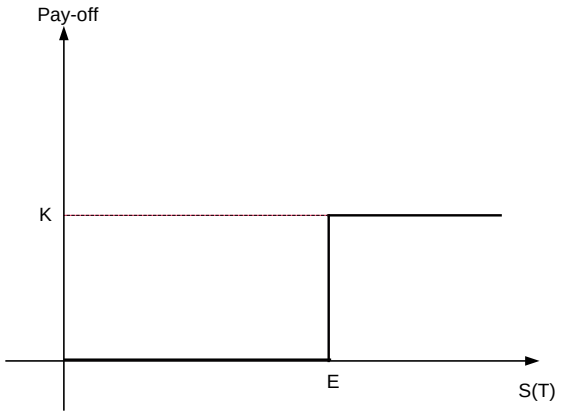

Figura 1.4: Pay-off di un'opzione digitale.

Anche per le opzioni con barriera esiste una esoticità notevole. Quelle più comuni si dividono in due famiglie: le *Knock in* e le *Knock out*. Nelle opzioni *Knock in* il pay-off finale viene corrisposto solo se il sottostante ha raggiunto un certo livello durante la vita dell'opzione; in altri termini sono opzioni che iniziano ad esistere solo se il sottostante colpisce una certa barriera.
Per contro le opzioni *Knock out* pagano un pay-off finale solo se il sottostante non ha mai toccato la barriera. In altri termini possiamo dire che le *Knock out* sono opzioni che cessano di esistere nel momento in cui il sottostante tocca la barriera.
Le opzioni con barriera sono sempre opzioni *over the counter*, ovvero opzioni che le società finanziarie creano per soddisfare le esigenze della propria clientela. Sono richieste in quanto hanno un prezzo più contenuto (avendo una clausola per la validazione o l'estinzione) rispetto alle corrispondenti opzioni plain vanilla.
Come esempi di opzioni *Knock in*, citiamo le:

- **opzioni down & in call (put):** che danno il pay-off di una plain vanilla call (put) se il sottostante scende almeno una volta sotto la barriera, altrimenti non pagano nulla;

- **opzioni up & in call (put):** che corrispondono il pay-off di un'opzione plain vanilla call (put) solo se il sottostante supera almeno una volta la barriera, diversamente non pagano nulla.

come esempi di opzioni *Knock out*, citiamo le:

- **opzioni down & out call (put):** che danno il pay-off di una plain vanilla call (put) solo se il sottostante non scende mai sotto la barriera, altrimenti non pagano nulla;

- **opzioni up & out call (put):** che corrispondono il pay-off di un'opzione plain vanilla call (put) solo se il sottostante non supera mai la barriera, diversamente non pagano nulla.

Esistono poi opzioni con doppia barriera di tipo *Knock out*, in cui viene corrisposto un pay-off solo se il sottostante rimane confinato tra le due barriere, oppure con doppia barriera di tipo *Knock in*, in cui il pay-off viene pagato solo se il sottostante, durante la vita dell'opzione, supera la barriera superiore oppure scende sotto quella inferiore. Una ulteriore esoticità viene ottenuta con le opzioni a cipolla ("onion option"). In quest'ultime opzioni si definiscono una successione di barriere inferiori $I_1 > I_2 > I_3 ..... > I_n$ e superiori $S_1 < S_2 < S_3 ..... < S_n$. Se durante la vita dell'opzione il sottostante rimane confinato nel corridoio più interno (ovvero se $I_1 < S(t) < S_1 \ \forall t \in (0,T)$) allora viene pagato a scadenza un premio P_1; se il sottostante supera una delle due barriere (L_1 o S_1) ma rimane confinato nel corridoio successivo ($I_2 < S(t) < S_2 \ \forall t \in (0,T)$) allora viene corrisposto un premio $P_2 < P1$; e così via. Se il sottostante non rimane confinato nemmeno nel corridoio più eterno (L_n, S_n) allora non viene corrisposto alcun pay-off. Questo tipo di opzioni a cipolla, può essere facilmente scomposto in una somma di opzioni con doppia barriera di tipo *Knock out*. In particolare è facile dimostrare che il pay-off definito sopra può essere riprodotto nella seguente maniera:

$$\text{Pay-off onion} = \sum_{i=1}^{N} \text{Pay-off double barrier}(I_i, S_i, P_i - P_{i-1}) \ . \qquad (1.84)$$

dove $P_0 = 0$ e Pay-off double barrier(I_i, S_i, P) è il pay-off di un'opzione *Knock out* avente barriera inferiore I_i, barriera superiore S_i e un premio a scadenza pari a P.

Un elemento importante che viene sempre specificato nei contratti, è ogni quanto verificare l'eventuale raggiungimento della barriera. Usualmente nella maggior parte dei casi, si considera il valore di chiusura del sottostante in ogni giorno della vita dell'opzione.
Come ultima considerazione, è interessante osservare che nelle opzioni con barriera l'assunzione di log-normalità per il processo che descrive il sottostante ha conseguenze ancora più importanti di quanto non avvenga per le plain vanilla. Infatti le opzioni con barriera entrano in funzione (o scompaiono) in seguito ad eventi estremi come il tocco della barriera. Se la distribuzione dei ritorni del sottostante è caratterizzata da code grasse, il tocco di una eventuale barriera è più probabile. Possiamo quindi aspettarci, ad esempio, che un'opzione *down & out* o *up & out* abbia un prezzo reale più basso di quanto non prescriverebbe il modello di BS (in cui il tocco di una barriera lontana dal valore attuale è un evento sottostimato); l'opposto per un'opzione *down & in* o *up & in*.

1.4.4 Opzioni cliquet e reverse cliquet

Un'altra tipologia di opzioni presente sul mercato, soprattutto quello francese, è dato dalle *cliquet option*. In queste opzioni il pay-off è definito nel seguente modo:

$$\text{Pay-off call cliquet} = \sum_{i=0}^{N-1} Max\left[S(t_{i+1}) - S(t_i), 0\right] \tag{1.85}$$

$$\text{Pay-off put cliquet} = \sum_{i=0}^{N-1} Max\left[S(t_i) - S(t_{i+1}), 0\right] \tag{1.86}$$

dove $S(t)$ è il sottostante, t_0 è la data attuale e $\{t_i\}_{i=1,...N}$ è un insieme di date future.
Il risultato ottenuto da queste opzioni è quello di bloccare il guadagno che si realizza con il passare del tempo. Consideriamo ad esempio una call cliquet su un indice S che al tempo iniziale valga 100, supponiamo di considerare tre date di rilevazione a 1, 2 e 3 anni. Se dopo il primo anno l'azione vale 110 l'opzione avrà maturato un guadagno di 10 (che pagherà a scadenza). A questo punto è come se lo strike della nostra call option venisse ri-fissato a 110. Alla fine del secondo anno se il sottostante chiude a 105 non verrà maturato alcun guadagno ma lo strike verrà riportato a 105. Infine se al compimento del terzo anno l'azione arriva a 120, si avrà un guadagno di 15. In totale il pay-off corrisposto alla scadenza dell'opzione sarà: $10 + 0 + 15 = 25$. In altri termini possiamo dire che un'opzione cliquet può essere vista come la somma di una serie di opzioni plain vanilla, una per ogni intervallo (t_i, t_{i+1}), con uno strike price che di volta in volta assume un valore pari a: $S(t_i),\ i = 0, ..., N-1$.
Come si vede il vantaggio fondamentale offerto da questa tipologia di opzioni è quello di garantire all'acquirente un blocco nei guadagni via via realizzati. È chiaro che questo vantaggio supplementare, come tutti i diritti, verrà pagato corrispondendo un premio più alto al momento della sottoscrizione dell'opzione. Basta infatti osservare che:

$$Max\left[S(t_{i+1}) - S(t_i), 0\right] \geq S(t_{i+1}) - S(t_i)\ , \tag{1.87}$$

da cui: Pay-off call cliquet $\geq$ Pay-off call plain vanilla con strike $S(t_0)$. (Relazione analoga vale nel caso delle put.) Ciò conferma che il prezzo di una *cliquet option* è sempre più grande della corrispondente *at the money plain vanilla option*[19]

Una variante più complicata delle cliquet sono le opzioni *reverse cliquet*.

[19]Per *at the money plain vanilla option* intendiamo una opzione standard in cui lo strike price sia pari al valore dell'azione ad oggi.

In esse il pay-off è definito dalle seguenti formule:

$$\begin{aligned}\text{Pay-off reverse call cliquet} &= Max\left\{0, C - \sum_{i=0}^{N-1} Max\left[S(t_{i+1}) - S(t_i), 0\right]\right\} \\ \text{Pay-off reverse put cliquet} &= Max\left\{0, C - \sum_{i=0}^{N-1} Max\left[S(t_i) - S(t_{i+1}), 0\right]\right\}\end{aligned} \tag{1.88}$$

dove C è una costante predefinita.
In questo caso il gioco consiste nel partire da un pay-off potenziale molto alto (C), a cui via via si scalano le performance del sottostante (positive nel primo caso e negative nel secondo) maturate nei vari periodi.

Le opzioni cliquet, avendo un pay-off dipendente dalla storia del sottostante, sono un altro esempio di "path dependent option".

1.4.5 Opzioni lookback

Le opzione *lookback* possono essere di tipo *call* o *put*. Il pay-off, per queste opzioni, viene definito in accordo alle seguenti formule:

$$\begin{aligned}\text{Pay-off call lookback} &= Max\left\{S(T) - Min\left[S(t)\right]_{t\in(0,T)}, 0\right\} \\ \text{Pay-off put lookback} &= Max\left\{Max\left[S(t)\right]_{t\in(0,T)} - S(T), 0\right\}\end{aligned} \tag{1.89}$$

In altri termini la call (put) lookback option è simile ad una *call (put) plain vanilla* in cui però lo *strike price* non è fissato all'istante iniziale ma è dato dal valore minimo (massimo) raggiunto dal sottostante durante la vita dell'opzione. Un elemento importante in questo tipo di opzioni è perciò la frequenza di osservazione del prezzo del sottostante. È infatti ovvio che rilevare il valore dell'azione giornalmente piuttosto che settimanalmente può portare ad un cambiamento nella determinazione del minimo. Nel caso in cui si supponga di adottare un'osservare continua, è possibile derivare delle formule di *pricing* esatte per questo tipo di opzioni.
Anche le *lookback option* forniscono un ulteriore esempio di opzioni *path dependent*, in quanto il pay-off non dipende sola dal valore finale raggiunto dal sottostante, ma anche dalla sua storia precedente.

1.4.6 Opzioni su basket

Per concludere la nostra breve panoramica sulle opzioni esotiche più diffuse, non possiamo non citare le opzioni scritte su basket di azioni. In questo caso la principale novità rispetto a tutte le tipologie viste fino ad ora, consiste nella presenza di più sottostanti. Illustrare una casistica completa delle

opzioni su basket trattate sul mercato è impossibile, ci limiteremo quindi a fornire alcuni esempi.

1.4.6.a Opzioni best asset

Nelle opzioni *best asset* il pay-off finale è dato dal migliore (o peggiore) pay-off scritto su ciascuna azione. Indichiamo con $\{S_i\}_{i=1,\dots,N}$ un set di azioni e per ciascuna di queste definiamo un pay-off$_i$ (usualmente la definizione è la medesima per ogni asset S_i). Il pay-off dell'opzione *best asset* è definito come:

$$\text{Pay-off} = Max\left[\text{Pay-off}_i\right] \ , \tag{1.90}$$

mentre per l'opzione *worst asset* è:

$$\text{Pay-off} = Min\left[\text{Pay-off}_i\right] \ , \tag{1.91}$$

A titolo di esempio illustriamo due tipici casi.

Consideriamo due sottostanti, S_1 e S_2; un esempio di opzione best asset sui due sottostanti è data da:

$$\text{Pay-off} = Max\left\{Max\left[\frac{S_1(T)}{S_1(t_0)} - 1, 0\right], Max\left[\frac{S_2(T)}{S_2(t_0)} - 1, 0\right]\right\} \ , \tag{1.92}$$

dove t_0 è l'istante iniziale e T è la maturity dell'opzione.
In questo caso abbiamo scelto come formula di pay-off sul singolo sottostante, quella di un'opzione *call plain vanilla*. Si noti anche che l'opzione *best asset* su S_1 e S_2 varrà più della migliore tra le opzioni *plain vanilla* scritte rispettivamente su S_1 e S_2.

Un altro esempio, che si incontra abbastanza spesso sul mercato, è quello di un'opzione che paga un pay-off pari a K se durante la vita dell'opzione nessuna delle azioni del basket ha una performance che scenda al di sotto di una certa percentuale p. In questo caso si tratta di un'opzione worst asset, con un pay-off sulla singola azione di tipo *knock out*:

$$\text{Pay-off}_i = \begin{cases} K & \text{se } \frac{S_i(t)}{S_i(0)} > p \ \ \forall t \in (0,T) \\ 0 & \text{diversamente} \end{cases}$$

1.4.6.b Opzioni su indici

Molte volte si incontrano delle opzioni definite su basket di azioni in cui il pay-off viene specificato utilizzando unicamente un indice costruito sul basket. Generalmente un indice su un basket di N azioni è definito assegnando: 1) a ciascuna azione il peso p_i $(0 \leq p_i \leq 1)$ che questa ha sull'indice stesso ad un certo istante iniziale t_0; 2) il valore dell'indice al tempo t_0: $I(t_0) = I_0$. Se consideriamo dunque l'indice composto da n_1 azioni S_1, n_2 azioni S_2 ecc. Avremo:

$$I(t) = \sum_{i=1}^{N} n_i \, S_i(t) \ , \tag{1.93}$$

imponendo la condizione sui pesi:

$$\frac{n_i\, S_i(t_0)}{I(t_0)} = p_i \ , \tag{1.94}$$

ne segue che:

$$I(t) = I(t_0) \sum_{i=1}^{N} p_i \, \frac{S_i(t)}{S_i(t_0)} \ . \tag{1.95}$$

Poiché, come abbiamo visto nel paragrafo 1.2.6, si assume generalmente che la dinamica stocastica seguita dalle azioni sia un processo di Wiener generalizzato, è facile mostrare che anche la combinazione lineare che definisce $I(t)$, segue la stessa dinamica. Se ne conclude che il *pricing* dell'opzione sul basket è ricondotto al caso di un'opzione su un singolo sottostante.

1.4.7 Definizione generale di opzione

Come si vede, sul mercato esistono una miriade di opzioni esotiche diverse e l'esposizione fornita sopra rappresenta solo la descrizione di quelle più comuni.
In generale possiamo dare la seguente definizione di opzione:
Def. Siano dati N sottostanti $i = 1, 2, 3.....N$. Sia $S_i(t)$ la funzione che assegna a ciascun sottostante i il suo valore al tempo t ($0 \leq t \leq T$). Un'opzione (di tipo cash) su questi sottostanti è un contratto finanziario tra due parti, una che vende l'opzione e l'altra che l'acquista:

- chi vende l'opzione riceve oggi una quantità di denaro prestabilita, detta premio;
- chi compra l'opzione acquisisce, in cambio del premio, un diritto, esercitabile unicamente in corrispondenza a certe date future: $t \in \{t_i\}_{i=1,2,....K}$. Questo diritto consiste nella possibilità di ricevere una quantità di denaro (al momento non definita e che potrà eventualmente essere nulla) pari a: $P = \mathcal{P}(S_1, S_2, ...S_N)$.

Dove $\mathcal{P}$ è un funzionale:

$$\begin{aligned} &\mathcal{P} : \mathcal{S}^N \rightarrow [0, \mathbb{R}] \ , \\ &\mathcal{S} = \{S : [0, t_i] \rightarrow [0, \mathbb{R}]\} \ . \end{aligned} \tag{1.96}$$

Se l'opzione è di tipo europeo allora $\{t_i\}_{i=1,2,....K} = \{T\}$, dove T è la *maturity date*. Se l'opzione è di tipo americano allora $\{t_i\}_{i=1,2,....K} = [0, T]$.

All'interno della definizione data sopra, ricadono come casi particolari le opzioni plain vanilla esaminate nel paragrafo 1.1.6. Ad esempio un'opzione

call europea sarà caratterizzata dal seguente funzionale che ne definisce il pay-off: $\mathcal{P}(S) = Max[S(T) - E, 0]$, dove E è lo *strike price*. In questo caso il funzionale $\mathcal{P}$ si riduce ad una semplice funzione, in quanto il *pay-off* dipende unicamente dal prezzo del sottostante a scadenza.
La definizione data sopra è abbastanza generale da includere tutte le opzioni esotiche generalmente trattate dal mercato. In particolare è facile verificare che tutti gli esempi considerati nei precedenti paragrafi rientrano un questo schema.

Mentre nel caso delle opzioni *plain vanilla* sono disponibili formule esatte per il *pricing*, in generale per le opzioni esotiche non è possibile trovare formule analitiche chiuse e si dovrà perciò ricorre a tecniche numeriche. A questo riguardo, uno degli strumenti più diffusi e potenti è il metodo Monte Carlo, che sarà l'argomento del prossimo capitolo.

1.4.8 Esercizi

Si consideri un'opzione digital *cash or nothing* di tipo call (vedi paragrafo 1.4.2) e si derivi una formula esatta per il prezzo.

di opzioni asiatiche

1.5 Una formula approssimata per la valutazione di opzioni asiatiche

In questa sezione deriveremo una formula chiusa approssimata per determinare il prezzo di un'opzione asiatica. Il metodo esposto consiste nell'approssimare la media dei prezzi di un'azione in corrispondenza delle diverse date di fixing, tramite una distribuzione log-normale. Tale approccio è stato sviluppato in letteratura da Turnbull e altri [15, 16]. Nel seguito verrà esposta una derivazione più vicina alle esigenze del mercato in cui si considerano un set di date discrete di fixing non necessariamente equispaziate, una curva dei tassi a priori non piatta e una volatilità non costante.
Consideriamo quindi un'opzione call con strike price E, di tipo asiatico, scritta su un'azione con valore iniziale $S(t_0)$. Non facciamo alcuna ipotesi sulla curva dei tassi (che potrà avere una sua struttura a termine) o sulla volatilità (che potrà anche essere variabile).
Il pay-off dell'opzione è definito come:

$$\text{Pay-off}_{\text{Asian}} = \text{Max}\left[\frac{1}{m+1}\sum_{i=0}^{m} S(t_i) - E, 0\right] , \tag{1.97}$$

dove $S(t_i)$ è il prezzo dell'azione sottostante in corrispondenza ai tempi t_i, $\{t_i\}_{i=0,1,.....m}$ è il set delle date di osservazione (includendo la data spot t_0), $T = t_m - t_0$ è il tempo a maturità e $m+1$ è il numero di date di fixing.
Ora il prezzo dell'azione al tempo t_j $(j = 1, \dots m)$, può essere scritto come:

$$S(t_j) = S(t_0)\, e^{\sum_{i=1}^{j}\left[\left(r_i - \frac{\sigma_i^2}{2}\right)\delta t_i + \sigma_i \sqrt{\delta t_i}\, e_i\right]} , \tag{1.98}$$

dove: e_i sono variabili stocastiche normali $N(0,1)$ (ovvero con media nulla e varianza unitaria); $\delta t_i = t_i - t_{i-1}$ è l'intervallo di tempo i-esimo; $r_i = r_{t_{i-1}\to t_i}$ e $\sigma_i = \sigma_{t_{i-1}\to t_i}$ sono rispettivamente il tasso di interesse e la volatilità del sottostante nell'intevallo (t_{i-1}, t_i).
Introduciamo ora le variabili stocastiche ausiliarie, $\{a_i\}$, definite ricorsivamente come segue:

$$\begin{aligned} a_0 &= \log\left(\frac{1 + m\, e^{a_1}}{m+1}\right) , \\ a_i &= \left(r_i - \frac{\sigma_i^2}{2}\right)\delta t_i + \sigma_i \sqrt{\delta t_i}\, e_i + \log\left[\frac{1 + (m-i)\; e^{a_{i+1}}}{m+1-i}\right] , \end{aligned} \tag{1.99}$$

con la condizione "iniziale":

$$a_m = \left(r_m - \frac{\sigma_m^2}{2}\right)\delta t_m + \sigma_m \sqrt{\delta t_m}\, e_m . \tag{1.100}$$

È conveniente riscrivere a_i's come:

$$a_i = \rho_i + v_i \, \hat{e}_i \; , \tag{1.101}$$

dove ρ_i e v_i^2 sono la media e la varianza delle a_i e $\{\hat{e}_i\}$ sono delle nuove variabili stocastiche con media zero e varianza unitaria. Queste ultime, al contrario delle e_i's, sono strettamente correlate l'una all'altra e cosa più importante, le loro PDF (probability distribution function), indicate nel seguito come $P_{\hat{e}_i}$, sono non normali.

Utilizzando le definizioni (1.99) riportate sopra, le equazioni che governano l'evoluzione della media dei prezzi del sottostante, diventano:

$$\frac{1}{m+1-k} \cdot \sum_{i=k}^{m} S(t_i) = \begin{cases} S(t_{k-1}) \cdot e^{\rho_k + v_k \, \hat{e}_k} & \text{if } k \geq 1 \\ S(t_0) \quad \cdot e^{\rho_0 + v_0 \, \hat{e}_0} & \text{if } k = 0 \end{cases} \tag{1.102}$$

che sono l'analogo delle leggi di evoluzione standard per il prezzo di un'azione, dove al posto di una variabile normale standard figura un'opportuna variabile stocastica $\hat{e}_i$.
È immediato a questo punto scrivere il prezzo dell'opzione, c_{asian}, come:

$$c_{\text{asian}} = e^{-r t_0 \to t_m \, T} \int_{-\infty}^{+\infty} \text{Max} \left[S(t_0) \, e^{\rho_0 + v_0 \, x} - E, 0 \right] \, P_{\hat{e}_0}(x) \, dx \, . \tag{1.103}$$

Perciò il prezzo di un'opzione asiatica, con date di fixing discrete, è ricondotto al calcolo di un integrale unidimensionale, l'esatto analogo di una semplice opzione call plain vanilla in cui al posto della classica funzione di probabilità gaussiana compare una PDF data da $P_{\hat{e}_0}$. ρ_0 e v_0 devono essere calcolate in accordo allo schema riportato sopra.

Cerchiamo ora di risolvere la relazione ricorsiva (1.99) considerando unicamente i primi due momenti (media e varianza), in altre parole le variabili stocastiche $\hat{e}_i$'s vengono trattate come delle pure variabili gaussiane.
È conveniente prendere l'esponenziale di entrambi i lati dell'equazione (1.99):

$$\begin{cases} (m+1) \, e^{a_0} = 1 + m \, e^{a_1} \; , & \text{if } i = 0 \\ (m+1-i) \, e^{a_i} = e^{(r_i - \frac{\sigma_i^2}{2}) \, \delta t_i + \sigma \sqrt{\delta t_i} \, e_i} \cdot \left[1 + (m-i) \; e^{a_{i+1}} \right] \; , & \text{if } i > 0 \end{cases} \tag{1.104}$$

a questo punto imponiamo che entrambi i lati dell'equazione riportata sopra abbiano la stessa media e varianza. Tenendo conto che[20]:

$$< f(e_i) \cdot f(a_{i+1}) > = < f(e_i) > < f(a_{i+1}) > \; , \tag{1.105}$$

prendendo il logaritmo di entrambi i lati dell'equazione e considerando le seguenti due approssimazioni:

[20] Infatti va ricordato che le variabili stocastiche e_i e a_{i+1} sono tra loro indipendenti e quindi il valore di aspettazione $< f(e_i) \cdot f(a_{i+1}) >$ può essere fattorizzato

- assumiano che la distribuzione per la media dei prezzi sia log-normale (ovvero che le $\hat{e}_i$ siano normali), allora $\langle e^{a_i} \rangle = \langle e^{\rho_i + v_i \hat{e}_i} \rangle \approx e^{\rho_i + \frac{v_i^2}{2}}$;
- linearizziamo le equazioni ricorsive al primo ordine in ρ_i and v_i^2;

otteniamo due insiemi di relazioni ricorsive (una per la media e l'altra per la varianza):

$$\begin{cases} \rho_0 + \frac{v_0^2}{2} = \frac{m}{m+1} \left(\rho_1 + \frac{v_1^2}{2} \right) , & \text{if } i = 0 \\ \rho_i + \frac{v_i^2}{2} = r_i \, \delta t_i + \frac{m-i}{m-i+1} \left(\rho_{i+1} + \frac{v_{i+1}^2}{2} \right) , & \text{if } i > 0 \end{cases} \tag{1.106}$$

$$\begin{cases} v_0^2 = \left(\frac{m}{m+1} \right)^2 v_1^2 , & \text{if } i = 0 \\ v_i^2 = \sigma_i^2 \, \delta t_i + \left(\frac{m-i}{m-i+1} \right)^2 v_{i+1}^2 , & \text{if } i > 0 \end{cases} \tag{1.107}$$

le quali devono soddisfare le seguenti condizioni "iniziali":

$$\begin{aligned} \rho_m &= (r_m - \frac{\sigma_m^2}{2}) \, \delta t_m , \\ v_m^2 &= \sigma_m^2 \, \delta t_m . \end{aligned}$$

Il sistema (1.106-1.107) può essere risolto analiticamente tramite una ricorsione all'indietro, andando da $i = m$ a $i = 0$. Il risultato finale è:

$$\rho_0 + \frac{v_0^2}{2} = \sum_{i=1}^{m} r_i \left(1 - \frac{i}{m+1} \right) \delta t_i , \tag{1.108}$$

$$v_0^2 = \sum_{i=1}^{m} \sigma_i^2 \left(1 - \frac{i}{m+1} \right)^2 \delta t_i , \tag{1.109}$$

Perciò il valore dell'opzione dato dall'equazione (1.103) si riduce alla classifa formula di Black & Scholes con una volatilità e un dividend yield aggiustati nel seguente modo:

$$d_{adj} = \frac{1}{T} \sum_{i=1}^{m} r_i \frac{i}{m+1} \, \delta t_i , \tag{1.110}$$

$$\sigma_{adj}^2 = \frac{1}{T} \sum_{i=1}^{m} \sigma_i^2 \left(1 - \frac{i}{m+1} \right)^2 \delta t_i . \tag{1.111}$$

Dal risultato ottenuto sopra è possibile derivare delle formule più semplici valide nel caso di:

- intervalli equispaziati per le date di fixing ($\delta t_i = t_i - t_{i-1} = T/m$),
- curva dei tassi piatta ($r_i = r$),

- volatilità costante ($\sigma_i = \sigma$),

ovvero:

$$d_{adj} = \frac{r}{2} \,, \tag{1.112}$$

$$\sigma_{adj}^2 = \frac{1}{3}\,\frac{2m+1}{2m+2}\,\sigma^2 \,. \tag{1.113}$$

Infine, le equazioni (1.110) e (1.111) diventano nel limite di rilevazioni continue (cioè $m \to \infty$):

$$d_{adj} = \frac{1}{T}\int_{t=0}^{T} r(t_0+t)\,\frac{t}{T}\,dt \,, \tag{1.114}$$

$$\sigma_{adj}^2 = \frac{1}{T}\int_{t=0}^{T} \sigma(t_0+t)^2\,\left(1-\frac{t}{T}\right)^2\,dt \,. \tag{1.115}$$

Le relazioni (1.110) e (1.111) riportate sopra, hanno una semplice interpretazione intuitiva dal punto di vista finanziario: l'equazione (1.110) indica che il calcolo della media ha l'effetto di ridurre il fattore di drift standard (ad esempio nel caso di curva dei tassi piatta e date di monitoraggio equispaziate, il dividend yield aggiustato è legato al tempo medio a scadenza, ovvero $T/2$). L'equazione successiva, relativa a σ_{adj}, rappresenta la riduzione nella volatilità effettiva dovuta al meccanismo della media nel prezzo del sottostante (in effetti è ben noto che le opzioni asiatiche sono caratterizzate da una volatilità più bassa, e quindi da un prezzo inferiore, rispetto alle corrispondenti opzioni plain vanilla). È infine interessante osservare che i valori di volatilità che caratterizzano il sottostante nei primi intervalli di rilevazione, giocano un ruolo più importante nella definizione della volatilità aggiustata σ_{adj}.
Formule simili a quelle riportate sopra sono state ottenute in letteratura [15, 16], nel caso di opzioni asiatiche campionate in maniera continua (ovvero $m \to \infty$) con curva dei tassi piatta e volatilità costante, imponendo che la distribuzione della media dei prezzi del sottostante sia approssimabile tramite una lognormale. Altri risultati (vedi ad es. [17]) sono stati proposti per opzioni con date di fixing discrete, assumendo sempre una curva dei tassi ed una volatilità costanti, senza però derivare esplicitamente una formula chiusa. È perciò interessante sottolineare che il risultato approssimato riportato nelle equazioni (1.110) e (1.111), rappresenta una formula chiusa valida in presenza di fixing discreti[21], non necessariamente equispaziati e con condizioni di mercato realistiche (ovvero tassi di interesse con una vera struttura a termine e volatilità non costante).

[21] In effetti, in questo caso, la volatilità aggiustata, σ_{adj}, può essede differente dal noto limite continuo, $1/\sqrt{3}\,\sigma$, specialmente quando si considerino poche date di rilevazione (vedi eq. (1.113)) oppure quando queste ultime non siano distribuite in modo uniforme.

Opzioni asiatiche: option pricing con differenti tecniche				
	$\sigma = 5$ %	$\sigma = 10$ %	$\sigma = 30$ %	$\sigma = 50$ %
(log-normal approx.)	4.274	4.871	8.742	12.913
ME ($l = 4$)	4.30799	4.90899	8.79859	12.96664
ME ($l = 6$)	4.30798	4.90899	8.80149	12.97995
ME ($l = 8$)	4.30798	4.90899	8.80142	12.98102
ME ($l = 10$)	4.30798	4.90899	8.80153	12.98124
ME ($l = 20$)	4.30798	4.90899	8.80151	12.98097
MC	4.30795 +/- 0.00003	4.9089 +/- 0.00014	8.80095 +/- 0.00062	12.980 +/- 0.0012
RNI	4.308	4.909	8.801	12.980
MDA	4.309	4.911	8.811	12.979

Tabella 1.1: Prezzo di un opzione call asiatica per i seguenti parametri: $S(t_0) = 100$, $r = 9$ %, $T = 1$ year, $E = 100$, $m = 52$ (numero di intervalli) e $\delta t_i = T/m$. La tabella mostra un confronto tra differenti tecniche: approssimazione log-normale - soluzione analitica basata sulle eq. (1.110) e (1.111), ME - Espansione nei Momenti, l è il numero di momenti considerato (Airoldi [18]), MC - Monte Carlo (10^8 scenari con l'uso della variabile antitetica), RNI - Recursive Numerical Integration (Lim [19]) e MDA - Mixed Density Approximation (Lim [19]).

Nella tabella 1.1 viene riportato un confrontro tra il prezzo di un'opzione asiatica call ottenuto tramite l'approssimazione analitica riportata sopra e altre popolari tecniche numeriche. Come si può notare l'errore rispetto al risultato "esatto" è ragionevolmente piccolo (sempre inferiore a 1 %).

Capitolo 2

Metodi numerici per il pricing di opzioni

2.1 Metodo Monte Carlo

2.1.1 Introduzione

In questa sezione illustreremo una delle principali tecniche utilizzate in finanza per l'*option pricing*: il metodo Monte Carlo. Esamineremo inizialmente questa metodologia da un punto di vista generale, partendo da un caso molto semplice [20]. Successivamente vedremo come le simulazioni Monte Carlo possano essere applicate alla finanza. Vedremo anche quali sono i punti di forza di questo metodo e in quali situazioni sia applicabile. Alla fine del capitolo si discuterà di alcune tecniche per migliorare l'efficienza delle ordinarie simulazioni Monte Carlo.

2.1.2 Il metodo Monte Carlo e π

Il metodo Monte Carlo è giustamente considerato uno dei più potenti mezzi disponibili nell'ambito del calcolo numerico. A dispetto della sua fama, si basa su un'idea estremamente semplice. Per esporla è utile risalire all'origine del suo nome. Questo deriva da un gioco molto popolare tra i bambini del principato di Monaco (da cui appunto il nome di metodo Monte Carlo). In questo gioco i partecipanti tracciano sul terreno un cerchio iscritto in un quadrato, una volta bendatisi gli occhi, cominciano a lanciare a caso delle pietre. Al termine si contano quante pietre sono cadute nel cerchio e quante nel quadrato. Si noti che il rapporto tra questi due numeri fornisce chiaramente una stima della superficie del cerchio (o più precisamente di $\pi/4$). Questo semplice gioco, come ora spiegheremo, è l'essenza del metodo Monte Carlo.
Lanciare una serie di pietre all'interno del quadrato corrisponde, da un punto di vista astratto, a generare una coppia di numeri casuali (r_1 e r_2), ciascuno dei quali sia compreso tra -1 e 1 (sarà quindi sufficiente disporre di una routine numerica che generi numeri casuali tra loro scorrelati nell'intervallo $(-1, 1)$, richiamandola due volte[1]). Questa coppia cadrà all'interno del cer-

[1]Questo punto è in effetti estremamente delicato. La routine numeriche dei calcolatori sono in grado di generare delle sequenze che, per correttezza, sarebbe bene definire pseudo casuali. Infatti la sequenza di numeri casuali (ovvero scorrelati) che un computer è in grado di generare, viene ottenuta tramite un programma che, per definizione produce un output deterministico. In effetti ogni routine di numeri random, viene inizializzata con un seme iniziale. Partendo dallo stesso seme si ottiene la medesima successione di numeri pseudo casuali. Generalmente tutti i generatori di numeri pseudo casuali utilizzano una relazione ricorsiva $x_{n+1} = f(x_n)$ al fine di generare una sequenza random $\{x_n\}$. Poiché la memoria del computer è limitata, è abbastanza intuibile che dopo un certo periodo, la sequenza tenderà a ripetersi. Questo introduce un elemento di correlazione nella successione, in grado di alterare potenzialmente la validità delle nostre simulazioni. È quindi sempre buona norma accertarsi che una simulazione utilizzi solo una porzione molto ridotta del periodo con cui la sequenza di numeri pseudo casuali si ripresenta. Inoltre se vogliamo ripetere la simulazione sarà bene utilizzare sempre un seme diverso, onde evitare di riottenere esattamente gli stessi risultati.

chio se $r_1^2 + r_2^2 \leq 1$, o nè rimarrà al di fuori in caso contrario. Supponiamo di ripetere molte volte questo esperimento, andando ogni volta a stabilire se il punto cade o meno all'interno del cerchio. Indichiamo con A_N i punti all'interno del cerchio, dove N è il numero totale di campioni considerati (in figura 2.1, A_N è rappresentato dai punti rossi pieni, N è invece dato dalla somma dei punti rossi pieni e dei bianchi). Nel limite di grande N, il

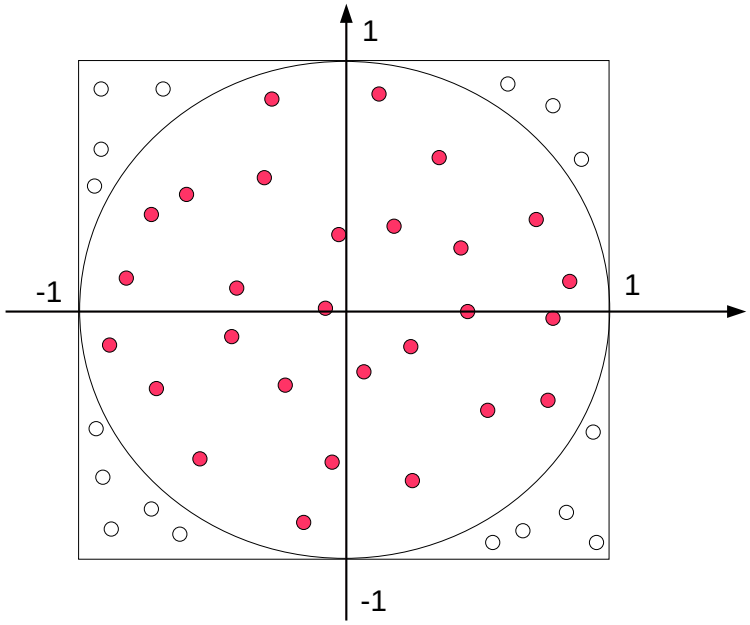

Figura 2.1: Monte Carlo per la stima di π.

rapporto A_N/N tende all'integrale:

$$\lim_{N\to\infty} A_N/N = \int_{(x,y)\in\mathcal{R}^2} f(x,y)\, p(x,y) dx\, dy\ . \tag{2.1}$$

dove:

$$f(x,y) = \begin{cases} 1 & \text{se } (x^2+y^2) \leq 1 \\ 0 & \text{altrimenti} \end{cases}$$

e

$$p(x,y) = \begin{cases} 1/4 & \text{se } x \in (-1,1) \text{ e } y \in (-1,1) \\ 0 & \text{altrimenti} \end{cases}$$

$p(x,y)$ rappresenta la densità di probabilità dei lanci (in questo caso, un valore uniforme all'interno del quadrato).
Come risulta chiaro dall'equazione (2.1), nel limite di grande N il rapporto A_N/N tende a $\pi/4$, fornendo così una stima di π. Essendo quello descritto un metodo statistico, il risultato sarà affetto da un'incertezza dell'ordine di $1/\sqrt{N}$, dove con N indichiamo il numero di campioni indipendenti. Questa

affermazione può essere facilmente provata ricorrendo al teorema del limite centrale. Questo afferma che:

Teorema Siano $\{w_i\}$ un set di variabili stocastiche indipendenti e identicamente distribuite. Supponiamo altresì che la distribuzione di probabilità, $p(w)$, che le caratterizza, abbia media μ e varianza, σ^2, ben definite. Se consideriamo la media $W = \frac{1}{N}\sum_{i=1}^{N} w_i$, questa è a sua volta una variabile stocastica che, nel limite $N \to \infty$, è descritta da una distribuzione gaussiana con media: μ e deviazione standard $\sigma/\sqrt{N}$.

Nel nostro caso ogni estrazione di coppie (x, y) ha una probabilità pari a $\pi/4$ di cadere all'interno del cerchio. Pertanto $f(x, y)$ è una variabile stocastica che assume unicamente due valori: 1 con probabilità $p = \pi/4$ e 0 diversamente. In base al teorema del limite centrale, il valor medio di f sarà una variabile distribuita gaussianamente con media pari a: $\pi/4$ e deviazione standard $\sqrt{p\,(1-p)/N}$. Nel caso in questione, questo risultato può essere ottenuto direttamente osservando che se ciascuna estrazione ha una probabilità di cadere all'interno del cerchio pari a $p = \pi/4$, dopo N lanci la probabilità di osservare n punti all'interno del cerchio sarà data da una distribuzione binomiale, la cui media è $N\,p$ e la cui varianza è: $N\,p\,(1-p)$. Riotteniamo perciò che il valore medio per la nostra stima è $\pi/4$ con un errore (standard deviation) pari a

$$\sigma_{\text{err.}} = \sqrt{\frac{p\,(1-p)}{N}}\ . \tag{2.2}$$

Come si vede l'errore commesso dipende da $1/\sqrt{N}$ tramite una costante di proporzionalità che nel caso specifico è data da:

$$\sqrt{p\,(1-p)} = \sqrt{\pi/4\,(1-\pi/4)} \approx 0.41\ . \tag{2.3}$$

Una dipendenza da $1/\sqrt{N}$ nella velocità di convergenza non è delle più rapide e certamente in $\mathbb{R}^2$, esistono algoritmi più efficienti per calcolare un integrale; ricordiamo a questo proposito il noto metodo del reticolo. Questo consiste nel decomporre il dominio di integrazione in piccoli quadrati di lato λ e nell'approssimare l'integrale tramite la somma: $\sum_{x_i} f(\vec{x_i})\lambda^2$, dove $\vec{x_i}$ è il generico punto al centro di ogni quadratino. Ovviamente il valore esatto si ottiene unicamente nel limite $\lambda \to 0$. Come vedremo fra poco, questo approccio risulta più efficiente del metodo Monte Carlo, almeno in $\mathbb{R}^2$. Le cose però cambiano nel momento in cui si considerino spazi a più alte dimensioni. In generale in uno spazio a D dimensioni, gli usuali algoritmi per il calcolo di un integrale, tendono a diventare inefficienti in quanto il tempo di calcolo cresce esponenzialmente al crescere di D e anche la complessità del problema tende ad aumentare. Viceversa il metodo Monte Carlo risulta facilmente estendibile ed inoltre la velocità di convergenza non dipende da

D. Supponiamo ad esempio di voler calcolare numericamente l'iper-volume di una sfera in D dimensioni. È chiaro che il metodo dei rettangolini diventa altamente inefficiente, mentre il metodo Monte Carlo risulta la scelta migliore. In tal caso per calcolare l'integrale in D dimensioni:

$$\int_{\vec{x}\in\mathcal{R}^D} f(\vec{x})\, p(\vec{x}) d\vec{x}\ . \tag{2.4}$$

dove:

$$f(\vec{x}) = \begin{cases} 1 & \text{se } \parallel \vec{x} \parallel \leq 1 \\ 0 & \text{altrimenti} \end{cases}$$

e

$$p(\vec{x}) = \begin{cases} 1/2^D & \text{se } x_j \in (-1,1)\ \forall j = 1, ..., D \\ 0 & \text{altrimenti,} \end{cases} ,$$

basterà generare $N\,D$ numeri casuali, raggruppandolo in N vettori, ciascuno di lunghezza D: $\vec{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, ..., x_D^{(i)})\ i = 1, ..., N$. A questo punto si utilizza la solita procedura algoritmica, in base alla quale si incrementa il contatore A_N di un'unità se e solo se $\parallel \vec{x}^{(i)} \parallel \leq 1$, dove l'indice i viaggia tra 1 e N. La grandezza A_N/N fornisce una stima del rapporto tra l'iper-volume della sfera e quello dell'ipercubo di lato 2, in D dimensioni. Ripetendo esattamente l'argomento visto prima, si dimostra che l'errore commesso scala come $K/\sqrt{N}$, dove il coefficiente moltiplicativo K dipende solamente dal rapporto tra l'iper-volume della sfera di raggio 1 e l'iper-cubo di lato 2 che la contiene.
Per contro, utilizzando il metodo del reticolo, possiamo immaginiamo di suddividere il dominio di integrazione in iper-cubi di lato λ; indicando con $\vec{x}^{(i)}$ il generico punto all'interno di ciascun iper-cubo, l'integrale D-dimensionale può essere riscritto come:

$$\begin{aligned} \int_D f(\vec{x}) d\vec{x} &= \sum_{\vec{x}^{(i)}} \int_{\vec{x}^{(i)}-\frac{\vec{\lambda}}{2}}^{\vec{x}^{(i)}+\frac{\vec{\lambda}}{2}} \Big[f(\vec{x}^{(i)}) + \sum_{j=1}^{D} \frac{\partial f}{\partial x_j}(\vec{x}^{(i)}) \left(x_j - x_j^{(i)}\right) + \\ &+ \frac{1}{2} \sum_{j=1}^{D} \sum_{k=1}^{D} \frac{\partial^2 f}{\partial x_j \partial x_k}(\vec{x}^{(i)}) \left(x_j - x_j^{(i)}\right) \left(x_k - x_k^{(i)}\right) + ... \Big]\, d\vec{x} = \\ &\approx \lambda^D \sum_{\vec{x}^{(i)}} f(\vec{x}^{(i)}) + \frac{1}{24}\, \lambda^{D+2} \sum_{\vec{x}^{(i)}} \sum_{j=1}^{D} \frac{\partial^2 f}{\partial x_j^2}(\vec{x}^{(i)})\ . \end{aligned} \tag{2.5}$$

Ne consegue che l'errore relativo commesso trascurando il termine al secondo ordine, è proporzionale a $\lambda^2 \sim N^{-\frac{2}{d}}$; in questo contesto, N rappresenta il numero di punti su cui valutare la funzione f (ovvero il numero di punti che formano il reticolo). Ne segue che il metodo del reticolo è vantaggioso rispetto al Monte Carlo solo per $d < 4$, mentre diventa via via più inefficiente in dimensioni maggiori. Si noti anche che l'estensione del metodo Monte Carlo

dal caso bidimensionale, $D = 2$, a quello generico in D dimensioni, è assolutamente banale. Riassumendo, la facilità di implementazione e di estensione del metodo Monte Carlo, unito al fatto che la velocità di convergenza risulta indipendente dalla dimensione dello spazio da campionare, fanno di questo algoritmo lo strumento ideale per affrontare il calcolo di integrali in spazi ad alte dimensioni.
L'esempio illustrato in questo paragrafo, per quanto semplicissimo, contiene già tutti gli elementi essenziali del metodo Monte Carlo. Astraendo dal presente caso, possiamo supporre di considerare un generico spazio $\mathcal{H}$ da cui sia possibile estrarre dei campioni x con densità di probabilità $p(x)$. Sia f una funzione definita su questo spazio a valori in $\mathbb{R}$. Allora:

$$\lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} f(x_i) = \int_{x\in\mathcal{H}} f(x)\, p(x) dx \ . \tag{2.6}$$

L'errore che commettiamo nel valutare il limite considerando solo un numero finito di campioni, è proporzionale a $1/\sqrt{N}$.

2.1.3 Miglioramenti al metodo Monte Carlo

Come abbiamo visto nel paragrafo precedente, una simulazione Monte Carlo è intrinsecamente un metodo statistico che non fornisce mai due volte lo stesso valore (almeno se il generatore di numeri casuali è inizializzato ogni volta con un seme diverso). In altri termini il metodo Monte Carlo fornisce una stima con un certo errore, che tende a scalare come $1/\sqrt{N}$. Ovviamente tanto più alta è la precisione che si desidera raggiungere tanto più dispendiosa sarà la simulazione (se ad esempio vogliamo raddoppiare la precisione nella stima di una grandezza, dobbiamo necessariamente quadruplicare il numero di campioni). Poiché in generale le simulazioni Monte Carlo sono piuttosto pesanti, sono state proposte una serie di miglioramenti al fine di ottenere stime più corrette senza aumentare eccessivamente i tempi di calcolo. In particolare esamineremo in dettaglio la tecnica della variabile di controllo.

2.1.3.a Tecnica della variabile di controllo

Una strada per migliorare la convergenza, consiste nel ridurre la varianza (e quindi l'errore) della stima fornita dalla simulazione Monte Carlo. Una metodologia, a questo riguardo molto utile e di facile implementazione, è la cosiddetta tecnica della variabile di controllo. Ne illustriamo i contenuti riprendendo l'esempio precedente sulla stima di π.
L'idea alla base di questa metodologia è quella di eseguire una simulazione Monte Carlo con l'obiettivo di stimare due grandezze: una sarà quella di nostro interesse mentre la seconda (detta variabile di controllo) è relativa ad un problema di cui sia nota la soluzione esatta. La valutazione Monte Carlo

per il problema di interesse verrà quindi corretta tramite la differenza tra la stima Monte Carlo della variabile di controllo ed il suo valore esatto (noto a priori). Vediamo nel concreto come questa tecnica possa essere utilizzata per migliorare la stima di π vista nel paragrafo precedente. Supponiamo di considerare un poligono di n lati inscritto nel cerchio di raggio unitario. Per ogni campione estratto (x, y), controlliamo non solo che questo cada o meno all'interno del cerchio ma anche all'interno del poligono (in particolare in figura 2.2, abbiamo considerato come poligono inscritto un quadrato. I punti che cadono dentro il cerchio ma fuori dal quadrato inscritto sono marcati in rosso pieno). In tal modo la nostra simulazione ci consentirà di stimare due

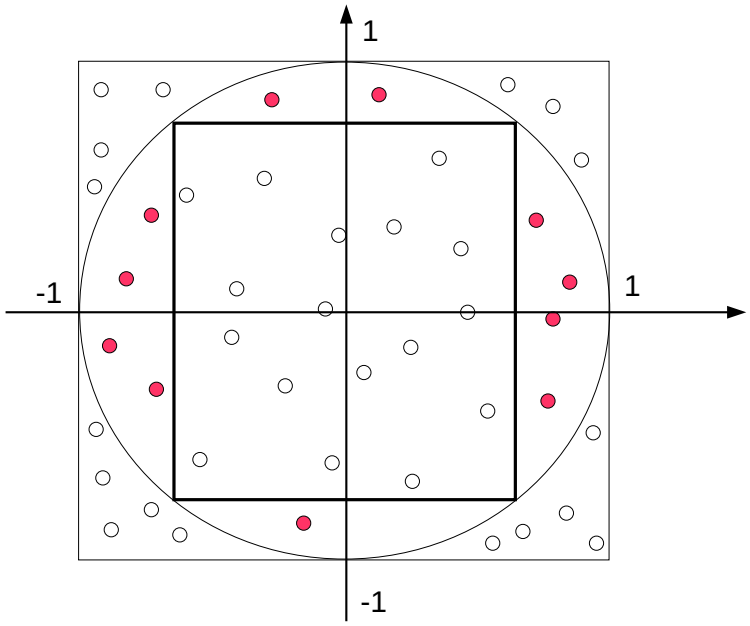

Figura 2.2: Monte Carlo per la stima di π con l'utilizzo di una variabile di controllo.

grandezze:

$$\frac{\text{Area cerchio di lato 1}}{\text{Area quadrato di lato 2}} = \frac{\pi}{4} = \lim_{N\to\infty} \frac{\#\text{ punti nel cerchio}}{N} \tag{2.7}$$

$$\frac{P_n^{\text{esatto}}}{\text{Area quadrato di lato 2}} = \lim_{N\to\infty} \frac{\#\text{ punti nel poligono}}{N} \tag{2.8}$$

nel primo caso supponiamo π incognito mentre nel secondo assumiamo come nota la formula chiusa per valutare l'area del poligono di n lati iscritto nel cerchio unitario:

$$P_n^{\text{esatto}} = n \sin\frac{\pi}{n}\sqrt{1-\sin^2\frac{\pi}{n}}\,. \tag{2.9}$$

Potremo quindi sostituire la nostra stima iniziale (2.7) con:

$$\frac{\pi}{4} \approx \frac{\#\text{ punti nel cerchio}}{N} + \frac{P_n^{\text{esatto}}}{4} - \frac{\#\text{ punti nel poligono}}{N} . \qquad (2.10)$$

Si noti che il valore d'aspettazione al secondo membro non è cambiato, e quindi nel limite di grande N convergerà ancora al valore $\pi/4$, ma la varianza della distribuzione si è ridotta. Questo perché abbiamo sottratto alla variabile, # punti nel cerchio/N, una variabile ad essa anticorrelata. Infatti se per una fluttuazione statistica si verifica una maggiore concentrazione di punti campionati all'interno del cerchio (rispetto alla sua superficie) allora è altamente probabile che la stessa cosa accada anche per il poligono. In tale situazione l'area del cerchio risulta sovrastimata ma questo eccesso viene controbilanciato dalla differenza tra l'area esatta del poligono meno la sua stima Monte Carlo. Ciò consente di ridurre gli errori statistici, pervenendo ad una migliore stima. La cosa può essere resa più rigorosa osservando che il secondo membro della (2.10) si può riscrivere come:

$$\frac{\pi}{4} \approx \frac{\#\text{ punti nel cerchio - }\#\text{ punti nel poligono}}{N} + \frac{P_n^{\text{esatto}}}{4} , \qquad (2.11)$$

dove l'unica variabile incerta è data appunto dal numero di punti che cadono all'interno dell'area compresa tra il cerchio e il poligono. Questa variabile è caratterizzata nel limite $N \to \infty$, da una distribuzione gaussiana con deviazione standard data dall'equazione (2.2), in cui questa volta p assume il valore:

$$p = \frac{\pi - P_n^{\text{esatto}}}{4} < \frac{\pi}{4} \qquad (2.12)$$

In altri termini l'errore viene ancora a dipendere da $1/\sqrt{N}$ ma con una costante di proporzionalità più piccola. In effetti questo risultato non deve stupire, in quanto nella stima Monte Carlo corretta tramite l'uso della variabile di controllo, si è introdotto un dato ulteriore: la conoscenza della superficie del poligono. Questa informazione supplementare ci consente di focalizzare la simulazione Monte Carlo unicamente sulla stima della parte di area rimanente. Ne consegue che l'errore statistico implicito nella stima (2.10) è tanto minore quanto più l'area del poligono (la variabile di controllo) tende a sovrapporsi (correlarsi) a quella del cerchio (in particolare tende a zero nel limite $n \to \infty$). Come conclusione possiamo dunque affermare che è sempre desiderabile che la variabile di controllo tenda ad "avvicinarsi" (ovvero a correlarsi) il più possibile alla variabile da stimare.

2.1.4 Il metodo Monte Carlo nel calcolo del prezzo di un'opzione europea

In questo capitolo vedremo come viene applicato il metodo Monte Carlo nell'ambito della finanza [21, 22]. In particolare focalizzeremo la nostra attenzione sul problema del calcolo del prezzo di una opzione europea.

2.1.4.a Formulazione del prezzo di un'opzione europea tramite "path integral"

Ricordiamo che il *pricing* di un'opzione consiste nell'andare a determinare quantitativamente il valore (cioè il prezzo) di un diritto, quello acquistato dal sottoscrittore dell'opzione.
Il problema del prezzo di un'opzione può essere riformulato, nel caso di un'opzione di tipo europeo, in termini di un path integral, come ora vedremo. Consideriamo un'opzione di tipo europeo. Sia $\mathcal{C}_{0,T}$ un generico cammino stocastico, in un mondo neutrale al rischio [2], che rappresenta una possibile evoluzione del sottostante a partire dal tempo iniziale $t = 0$ (in cui assume il valore iniziale S_0) fino al tempo T (data di scadenza dell'opzione). Sia $p(\mathcal{C}_{0,T})$ la densità di probabilità di questo cammino. Indichiamo infine con $\mathcal{P}(\mathcal{C}_{0,T})$, il pay-off calcolato sul cammino. Allora il prezzo dell'opzione sarà dato da:

$$\int_{\mathcal{C}_{0,T}} p(\mathcal{C}_{0,T})\, \mathcal{P}(\mathcal{C}_{0,T})\, e^{-rT} \ . \tag{2.13}$$

Il problema del pricing di un'opzione europea si riduce quindi al calcolo di un path integral su tutti i possibili cammini stocastici che abbiano inizio in S_0 al tempo $t = 0$.

2.1.4.b Il Monte Carlo nel pricing di un'opzione

L'aver ridotto il pricing di un'opzione europea ad un integrale, consente di applicare il metodo Monte Carlo alla stima del suo valore. Si noti infatti la similitudine tra l'integrale dell'eq. (2.13) e quello dell'eq. (2.6). Volendo fare un parallelo con il problema del calcolo dell'area del cerchio:

Confronto MC nel calcolo di π e nell'*option pricing*	
MC calcolo di π	**MC Option pricing**
calcolo area del cerchio	calcolo path integral per il pricing dell'opzione
densità di probabilità uniforme $p(x, y)$	densità di probabilità $p(\mathcal{C})$
coppia ordinata (x, y) con il vincolo $-1 \leq x \leq 1$ e $-1 \leq y \leq 1$	cammino stocastico $\mathcal{C} = \{S(t)\}_{t\in[0,T]}$ con il vincolo $S(t = 0) = S_0$
funzione $f(x, y)$ uguale a 1 all'interno del cerchio e 0 altrove	funzione per il pay-off dell'opzione $f(\mathcal{C})$

Tabella 2.1: Confronto tra la metodologia Monte Carlo per il calcolo di π e quella utilizzata nell'*option pricing*

[2] Porremo quindi $\mu = r$. Questa assunzione, che è giustificabile in base al principio di valutazione neutrale al rischio, semplifica enormemente il problema

Come si vede cambiano la complessità del problema e le dimensioni dello spazio che dobbiamo campionare ma non la "ricetta" da seguire. Il metodo Monte Carlo applicato all'option pricing consisterà quindi nei seguenti passi:

- Campionare in maniera diretta lo spazio dei cammini stocastici $S(t)$ con il vincolo $S(t=0) = S_0$. Per far questo si dovranno seguire i seguenti punti:

  (A) Si dividerà l'intervallo $(0, T)$ in L intervallini discreti ciascuno di lunghezza $\Delta t = T/L$. Idealmente Δt dovrebbe essere reso molto piccolo.

  (B) Si costruirà quindi un cammino stocastico procedendo nella seguente maniera: si estrae un campione casuale w da una distribuzione gaussiana con media nulla e varianza unitaria; si definisce quindi il nuovo valore di S al tempo $t_1 = 0 + \Delta t$ come:

  $$S(t_1) = S_0 + S_0\, r\, \Delta t + S_0\, \sigma\, \sqrt{\Delta t}\, w\ . \tag{2.14}$$

  dove w è un numero pseudo casuale, estratto da una distribuzione normale, $\phi(0,1)$ con media nulla e varianza unitaria.
  A questo punto si ripetono iterativamente i passi riportati sopra, estraendo un nuovo campione dalla distribuzione $\phi(0,1)$ e costruendo il nuovo valore di S al tempo t_2. Questa procedura ricorsiva, avrà termine dopo L passi.
  In tal modo abbiamo campionato un cammino stocastico con drift pari a r e volatilità σ, che soddisfa il vincolo $S(t=0) = S_0$.

  (C) Il punto (B) viene ripetuto N volte. In tal modo si ottengono N cammini $\{\mathcal{C}_i\}_{i=1,\ldots,N}$ che avranno automaticamente la corretta densità di probabilità $p(\mathcal{C})$.

- Si valutare il funzionale $\mathcal{P}$ su ciascun cammino $\mathcal{C}_i$;

- Infine si calcola la media: $\frac{1}{N}\sum_{i=1}^{N}$ Pay-off$(\mathcal{C}_i)$, e la si sconta ad oggi utilizzando il tasso privo di rischio. Quest'ultimo valore fornisce una stima del prezzo dell'opzione.

2.1.4.c Generazione di numeri pseudo casuali distribuiti normalmente

Un punto fondamentale nella sequenza di passi illustrata sopra, è la generazione di numeri casuali, w, estratti da una distribuzione gaussiana con media nulla e varianza unitaria (punto (B)). In generale molte librerie numeriche mettono già a disposizione dei generatori che consentono di campionare direttamente questo tipo di distribuzione. Può però accadere che

si abbia a disposizione unicamente un generatore di numeri pseudo casuali, con distribuzione uniforme nell'intervallo $(0, 1)$. In questi casi è possibile fare ricorso a delle semplici trasformazioni che consentono di ottenere numeri casuali soggetti ad una distribuzione di probabilità gaussiana. Supponiamo dunque che p e q siano due variabili stocastiche scorrelate, distribuite uniformemente nell'intervallo $(0, 1)$. È possibile dimostrare che le due variabili derivate:

$$\begin{aligned} x &= \sqrt{-2\log p}\,\sin(2\pi q)\,, \\ y &= \sqrt{-2\log p}\,\cos(2\pi q)\,, \end{aligned} \tag{2.15}$$

sono fra loro scorrelate e distribuite normalmente con media nulla e varianza unitaria. Il principale inconveniente di questa trasformazione sta nel fatto che, da un punto di vista computazionale, il calcolo di funzioni trigonometriche risulta sempre oneroso. Più vantaggioso, a tal riguardo, risulta la seguente trasformazione:

$$\begin{aligned} w &= s\sqrt{-2\frac{\log u}{u}}\,, \\ z &= t\sqrt{-2\frac{\log u}{u}}\,, \end{aligned} \tag{2.16}$$

dove questa volta s e t sono due variabili distribuite uniformemente nell'intervallo $(-1, 1)$ e $u = s^2 + t^2$, deve soddisfare il vincolo: $u \leq 1$ (ovvero (s, t) devono rappresentare le coordinate di un punto racchiuso all'interno del cerchio di raggio unitario). Anche in questo caso si può dimostrare che w e z sono due variabili gaussiane scorrelate. Poiché da un punto di vista numerico il calcolo di radici quadrate e logaritmi è meno dispendioso rispetto al caso di funzioni trigonometriche, la trasformazione (2.16) è preferibile alla (2.15). L'inconveniente principale presente in (2.16), risiede nel fatto che per ogni coppia di numeri casuali scorrelati, $(s, t) \in [-1, 1]\text{x}[-1, 1]$, dobbiamo preventivamente accertarci che valga la condizione $s^2 + t^2 \leq 1$, viceversa si deve scartare la coppia. Poiché la probabilità che la generica coppia (s, t) cada all'interno del cerchio unitario è pari a $\pi/4$, l'efficienza di questo processo sarà del 78.5%.
Le due trasformazioni riportate sopra costituiscono il cosiddetto metodo di Box–Muller.

2.1.4.d Simulazione Monte Carlo in presenza di più sottostanti

Nel paragrafo 2.1.4.b, abbiamo esaminato la situazione in cui vi sia solamente un sottostante che segua un processo log-normale (il cosiddetto modello browniano univariato). In generale però può essere necessario considerare più sottostanti (ad esempio nel caso di opzioni su basket di azioni), in tal

caso si fa riferimento ad un modello browniano multivariato. In questo modello oltre a definire quale sia la volatilità σ_i (ovvero la deviazione standard) dei ritorni di ogni singola azione S_i, è necessario definire anche l'insieme delle correlazioni [3], $\rho_{i,j}$, tra i ritorni di ogni coppia di azioni S_i, S_j. Si definisce in questo modo, quella che prende il nome di matrice di varianza covarianza, in cui nella diagonale principale figurano le volatilità mentre gli elementi fuori diagonale rappresentano le correlazioni:

$$\mathbf{C} = \begin{pmatrix} \sigma_1 & \rho_{1,2} & \cdots \\ \rho_{2,1} & \sigma_2 & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

La matrice C è ovviamente una matrice simmetrica in cui $C_{i,i} \geq 0$ e $-1 \leq C_{i,j} \leq 1 \ \ \forall i \neq j$.
Per simulare i cammini Monte Carlo seguiti dalle azioni $\{S_i\}$ dobbiamo tener conto anche delle loro correlazioni. Consideriamo il caso molto semplice in cui vi siano solamente due azioni ($N = 2$). Per generare una successione di variabili r_1 e r_2 che abbiano tra loro correlazione $\rho_{1,2}$ e volatilità σ_1 e σ_2 rispettivamente, consideriamo due variabili stocastiche scorrelate x_1 e x_2, estratte da una distribuzione normale a media nulla e varianza unitaria, $\Phi(0, 1)$. Consideriamo quindi la seguente combinazione lineare di x_1 e x_2:

$$\begin{aligned} r_1 &= \sigma_1 \, x_1 \\ r_2 &= \sigma_2 \left(\rho \, x_1 + \sqrt{1 - \rho^2} \, x_2 \right) \ . \end{aligned} \tag{2.17}$$

È facile dimostrare che:

$$\begin{aligned} &\langle r_1 \rangle = \langle r_2 \rangle = 0 \ , \\ &\langle r_1^2 \rangle = \sigma_1 \ , \\ &\langle r_2^2 \rangle = \sigma_2 \ , \\ &\mathrm{corr}(r_1, r_2) = \rho \ . \end{aligned}$$

Di conseguenza partendo da due variabili stocastiche scorrelate a varianza unitaria, è pressoché immediato ottenere le variabili stocastiche r_1 e r_2 con le varianze e la correlazione desiderate. Questo è un risultato molto importante in quanto generalmente le routine di calcolo sono in grado di produrre, in

[3] Ricordiamo che la correlazione tra due serie $\{x_i\}$ e $\{y_i\}$ è definita come:

$$\mathrm{corr}(x, y) = \frac{\sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y})}{\left[\sum_{i=1}^{n} (x_i - \overline{x})^2 \sum_{i=1}^{n} (y_i - \overline{y})^2 \right]^{1/2}}$$

maniera molto semplice, dei campioni casuali estratti da una distribuzione normale $\Phi(0,1)$. Utilizzando questi campioni (che sono tra loro scorrelati) è possibile, tramite la procedura sopra descritta, generare coppie di sequenze stocastiche tra loro correlate.
La metodologia illustrata sopra può essere poi estesa al caso generico di n variabili.

2.1.4.e Miglioramenti al metodo Monte Carlo nell'option pricing

In questo paragrafo esamineremo alcune semplici tecniche che consentono di ridurre notevolmente l'errore statistico insito nelle valutazioni Monte Carlo, senza incrementare il numero di cammini utilizzati.

Miglioramenti nella discretizzazione del processo stocastico Un primo importante miglioramento che si può apportare allo schema del paragrafo (2.1.4.b), riguarda la discretizzazione del processo stocastico.
I passi A e B delle procedura per implementare il Monte Carlo, sono validi in generale per qualunque processo stocastico si consideri al fine di modellizzare l'evoluzione del sottostante. La discretizzazione dell'intervallo di tempo $(0,T)$, in tanti intervallini di lunghezza $\Delta T = T/L$ e la successiva rappresentazione del cammino $\mathcal{C}$ tramite una curva spezzata a tratti (2.14), risulterà tanto più precisa quanto maggiore è il numero di intervalli considerati. Idealmente si dovrebbe far tendere L a infinito. Ovviamente nella pratica si dovrà sempre fare riferimento ad un numero finito di passi e questo introduce un errore sistematico nel calcolo. La cosa risulta particolarmente evidente osservando che il valore di S_i all'i-esimo step potrebbe in linea di principio diventare anche negativo, in contraddizione con il fatto che: 1) da un punto di vista finanziario non ha senso parlare di prezzi negativi per un'azione e 2) dall'equazione (1.36) risulta che i valori possibili per la variabile S su qualunque orizzonte temporale, sono sempre positivi.
A questo problema, sollevato dalla procedura di discretizzazione utilizzata, si può rimediare osservando che nel caso di un processo log-normale, si ha una formula analitica esatta per descrivere la distribuzione di probabilità che caratterizza il prezzo del sottostante. Infatti utilizzando la (1.36) possiamo sostituire l'equazione (2.14) con:

$$S(t_i) = S(t_{i-1})\, e^{\left(r-\frac{\sigma^2}{2}\right)\Delta t+\sigma\sqrt{\Delta t}\, w}\,, \tag{2.18}$$

Questo consente di eliminare l'errore sistematico che deriva dalla discretizzazione. Ovviamente ciò risulta possibile unicamente quando il processo stocastico che descrive il sottostante può essere risolto esattamente. In mancanza di questo si dovrà necessariamente ricorrere alla formula (2.14).
Si noti che nell'equazione (2.18), come del resto nella (2.14), abbiamo sostituito a μ il tasso risk free r. Questo perché la valutazione del prezzo dell'opzione va sempre condotta in un mondo neutrale al rischio.

Metodo della variabile antitetica nell'option pricing Questa metodologia è particolarmente utilizzata nell'ambito dell'*option pricing* in quanto consente di ridurre gli errori statistici delle simulazioni Monte Carlo, apportando una variazione minima allo schema descritto nel paragrafo (2.1.4.b). Come abbiamo visto, per generare un cammino stocastico $\mathcal{C}$, dobbiamo estrarre L numeri casuali da una distribuzione gaussiana univariata (L rappresenta il numero di intervalli in cui si è divisa la durata temporale dell'opzione). Sia $\{w_i\}$, l'insieme di questi numeri pseudo casuali e sia $\mathcal{C}_{\{w_i\}}$ il cammino stocastico che ne risulta. Senza rigenerare una nuova sequenza, possiamo considerare il cammino stocastico speculare a $\mathcal{C}$ che si ottiene considerando il set: $\{-w_i\}$. Questo corrisponde ad un secondo cammino $\hat{\mathcal{C}}$ che risulta fortemente anticorrelato rispetto al primo (infatti le oscillazioni stocastiche di $\hat{\mathcal{C}}$ sono opposte a quelle di $\mathcal{C}$). Se ora generiamo N coppie di questi cammini gemelli e calcoliamo il prezzo dell'opzione, otterremo una stima con un errore statistico più basso. La ragione risiede nel fatto che siamo riusciti a campionare più efficacemente lo spazio del nostro problema, avendo considerato cammini maggiormente scorrelati tra di loro. La cosa risulta ovvia osservando che:

- indicato con $P_{\mathcal{C}}$ il pay-off dell'opzione sul cammino $\mathcal{C}$;
- indicato con $P_{\hat{\mathcal{C}}}$ il pay-off dell'opzione sul cammino $\hat{\mathcal{C}}$;
- indicato con $P_{\mathcal{C}+\hat{\mathcal{C}}} = \frac{P_{\mathcal{C}}+P_{\hat{\mathcal{C}}}}{2}$ la media tra i due pay-off;
- indicato con ϵ l'errore statistico su $P_{\mathcal{C}}$ e con ρ la correlazione tra $P_{\mathcal{C}}$ e $P_{\hat{\mathcal{C}}}$;

allora l'errore statistico su P, dopo aver generato N coppie di cammini, è:

$$\text{err. P} = \frac{\epsilon\sqrt{\frac{(1+\rho)}{2}}}{\sqrt{N}} < \frac{\epsilon}{\sqrt{N}} \ . \tag{2.19}$$

si ottiene così una riduzione dell'errore statistico, diminuzione che è tanto maggiore quanto più i due cammini gemelli sono anticorrelati. Non cambia invece la dipendenza dall'inverso della radice quadrata del numero di cammini.
Si noti infine che dal punto di vista computazionale, generare il secondo cammino a partire dal primo ha un costo pressoché nullo. Si ha così un doppio beneficio.

Metodo della variabile di controllo nell'option pricing È utile rivedere il metodo della variabile di controllo, nel caso del pricing di un'opzione tramite metodo Monte Carlo. Come abbiamo visto nel paragrafo 2.1.3.a, il metodo della variabile di controllo consiste nell'eseguire una simulazione

Monte Carlo su due variabili: una ha un valore ignoto di cui vogliamo fornire una stima, mentre per l'altra è disponibile una formula chiusa per la sua valutazione. La seconda prende il nome di variabile di controllo. Il metodo consiste nel correggere la stima Monte Carlo per la variabile ignota tramite la differenza (o il rapporto) tra il valore esatto della variabile di controllo e la sua stima Monte Carlo. Se indichiamo il prezzo Monte Carlo di un'opzione esotica A con P_A^{MC} e consideriamo un'opzione di controllo B, il cui prezzo fornito dallo stesso Monte Carlo sia P_B^{MC}, mentre il suo valore esatto sia P_B^{VERO}, allora possiamo ottenere una migliore stima di P_A tramite la formula:

$$P_A^{MIGLIORATO} = P_A^{MC} + P_B^{VERO} - P_B^{MC} \ . \tag{2.20}$$

Questo metodo sarà tanto più efficace quanto più la variabile di controllo tende ad essere simile alla variabile incognita (si rammenti il caso del poligono iscritto nel cerchio). Nel caso dell'option pricing, è perciò utile scegliere un'opzione (di cui sia nota una formula chiusa per il prezzo) con un pay-off molto simile a quello dell'opzione sotto indagine. Ad esempio se vogliamo calcolare tramite un Monte Carlo il prezzo di un'opzione call down & out con barriera, la scelta migliore sarà quella di utilizzare come variabile di controllo un'opzione call con uguali caratteristiche (ovvero, stesso *strike price*, stessa *maturity* ecc.).
Come nel caso dei poligoni e del cerchio (vedi l'esempio del paragrafo 2.1.3.a), anche in questo caso questo approccio non riduce la dipendenza dell'errore dal numero di campioni (ovvero l'errore commesso sarà ancora del tipo: $K/\sqrt{N}$) ma permette di ridurre il coefficiente moltiplicativo K.

2.2 Alberi Binomiali

In questo capitolo esporremo, in sintesi, la metodologia numerica basata sugli alberi binomiali. In particolare introdurremo inizialmente tale tecnica in un contesto non finanziario (paragrafo 2.2.1) per poi passare al mondo della finanza in senso stretto (paragrafo 2.2.2). Nei paragrafi 2.2.3 e 2.2.4, presenteremo le due possibili versioni con cui gli alberi binomiali possono venire generati. Nel paragrafo 2.2.6 verrà descritto come effettuare la valutazione di un derivato ricorrendo alla generazione di alberi binomiali. Il capitolo si conclude con una valutazione dei vantaggi e degli svantaggi di questa metodologia rispetto ad altre note tecniche numeriche, quali ad esempio il metodo Monte Carlo ed il metodo alle differenze finite (che verrà introdotto nel capitolo 2.3).

2.2.1 Gli alberi binomiali nel gioco d'azzardo - esercitazione

Si consideri il seguente problema: un giocatore d'azzardo vuole mettersi al riparo da eventuali perdite al casinò e stipula per questo un'assicurazione. Tale contratto prevede che il giocatore parta con un patrimonio iniziale C e possa effettuare un numero di puntate alla roulette pari a n giocate, puntanto di volta in volta sul rosso o sul nero una percentuale prefissata p del patrimonio a sua disposizione in quel dato momento. Al termine delle n puntate, se il capitale disponibile sarà inferiore al patrimonio iniziale C, il giocatore riceverà un indennizzo pari alla differenza tra C ed il valore attuale (ovvero verrà risarcito dell'eventuale perdita subita).
Si domanda qual'è il giusto prezzo che il cliente dovrà versare per questa polizza assicurativa.
Si effettui il calcolo trascurando la presenza del numero zero alla roulette (in altri termini la probabilità di ottenre rosso o nero è pari, in entrambi, i casi al 50 %).

2.2.2 Introduzione agli alberi binomiali in finanza

La tecnica degli alberi binomiali per il pricing è estemamente utile da un punto di vista didatto anche se, come metodo numerico, si rivela poco robusto, soprattutto se paragonata al metodo delle differenze finite (che, come vedremo, è strettamente imparentato con il metodo degli alberi trinomiali). Il metodo degli alberi binomiali consiste nell'approssimare il processo stocastico lognormale per le azioni (tipicamente continuo) tramite un processo discreto su reticolo. Quest'ultimo dovrà essere in grado di riprodurre il corretto limite continuo nel momento in cui il passo reticolare [4] venga fatto tendere a zero.

[4] Ovvero l'intervallo di tempo δt in cui si suddivide l'intervallo di tempo $(0, T)$, dove T è la "maturity" dell'opzione

In particolare, dividiamo il generico intervallo di tempo $(0, T)$ in n intervalli di ampiezza $\delta t = T/n$, e supponiamo di approssimare l'evoluzione stocastica dell'azione (che al tempo $t = 0$ assume il valore S_0) tramite una semplice evoluzione a due soli valori (al posto degli infiniti che l'azione può assumere in base al modello log-normale):

$$S(\delta t) = \begin{cases} S_0\, u & \text{movimento verso l'alto di S con probabilità } p \\ S_0\, d & \text{movimento verso il basso di S con probabilità } 1-p \end{cases} \tag{2.21}$$

dove u e d rappresentano i fattori di rialzo e ribasso, rispettivamente con probabilità p e $1 - p$.
L'albero binomiale viene quindi ottenuto re-iterarando la regola rappresentata dall'equazione (2.21) su più stadi, fino ad arrivare al tempo $t = n\,\delta t = T$. I fattori p, u e d devono essere scelti in modo da garantire che nel limite del continuo (ovvero quando $n \to \infty$ e quindi $\delta t \to 0$) il modello discreto converga al modello log-normale standard. È facile dimostrare che condizione necessaria e sufficiente perché questo avvenga è che la media e la varianza di $S(\delta t)$, relativamente al processo discreto binomiale, coincidano con quelle del processo continuo, ovvero valgano le seguenti due condizioni:

- in un mondo neutrale al rischio (e tale è il framework in cui dobbiamo operare per ottenere il prezzo corretto di un'opzione) il valore atteso che l'azione assume nell'intervallo temporale $(0, \delta t)$, deve essere pari all'incremento determinato dal tasso privo di rischio:

$$\langle S(\delta t) \rangle = puS_0 + (1-p)dS_0 = S_0\, e^{r\delta t}\ , \tag{2.22}$$

  da cui

$$p = \frac{e^{r\delta t} - d}{u - d}\ , \tag{2.23}$$

- la varianza di S sull'intervallo δt sia:

$$\langle S(\delta t)^2 \rangle = S_0^2\ \left[p\, u^2 + (1-p)\, d^2\right] = S_0^2\, e^{(2\, r + \sigma^2)\, \delta t}\ , \tag{2.24}$$

Le due condizioni riportate sopra (condizioni rispettivamente sulla media e sulla varianza della variabile stocastica $S(\delta t)$, non sono però sufficienti a determinare in maniera univoca le tre variabili p, u e d, questo implica che la costruzione di un albero binomiale in grado di riprodurre per $\delta t \to 0$ il corretto limite continuo (rappresentato dal modello log-normale) non è univoca. In effetti in letteratura sono state proposte due soluzioni (tra le infinite possibili); queste saranno l'oggetto dei prossimi due paragrafi.

2.2.3 Alberi Binomiali per un sottostante - metodo di Cox, Ross e Rubinstein (CRR)

Cox, Ross e Rubinstein, al fine di trovare una soluzione univoca, impongono un'ulteriore condizione che si affianca alle equazioni (2.23) e (2.24), ovvero:

$$u\,d = 1\ , \tag{2.25}$$

In tal modo l'albero che viene generato ha la proprietà di avere una serie di nodi esattamente pari a S_0, ovvero l'albero risulta simmetrico rispetto alla retta che passa per $S = S_0$ (vedi figura 2.3). La condizione (2.25), insieme

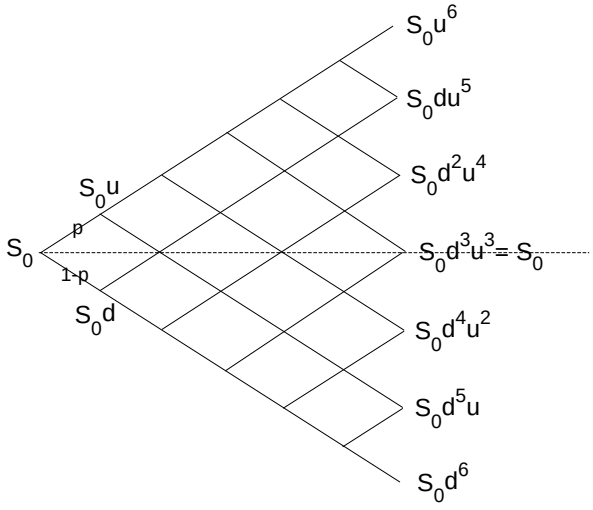

Figura 2.3: Albero binomiale CRR.

alle due equazioni (2.23) e (2.24), porta a determinare la seguente soluzione per u e d:

$$u = e^{\sigma\sqrt{\delta t}}\ , \tag{2.26}$$

$$d = e^{-\sigma\sqrt{\delta t}}\ . \tag{2.27}$$

2.2.4 Alberi Binomiali per un sottostante - metodo di Rubinstein

La seconda soluzione, dovuta a Rubinstein, può essere più utilmente illustrata richiamando direttamente l'equazione stocastica che esprime il valore aleatorio di $S(\delta t)$:

$$S(\delta t) = S(0)\, e^{\left(r-q-\frac{\sigma^2}{2}\right)\delta t+\sigma\sqrt{\delta t}\,z}\ , \tag{2.28}$$

q rappresenta il dividend yield dell'azione.
L'idea per derivare un modello discreto in grado di riprodurre nel limite del continuo la fomula riportata sopra, consiste essenzialmente nel sostituire la variabile aleatoria normale z con una variabile stocastica bimodale, $z^{(d)}$ che assuma con uguale probabilità i seguenti due valori:

$$z^{(d)} = \begin{cases} +1 & \text{con probabilità } 0.5 \\ -1 & \text{con probabilità } 0.5 \end{cases} \tag{2.29}$$

In tal modo si ottiene un albero binomiale, in cui la varianza e la media della variabile bimodale $z^{(d)}$ rimane identica a quella della variabile continua z (ovvero media nulla e varianza unitaria). Considerando un albero a più stadi molto fitto (ovvero in cui $n \to \infty$), il prezzo generico del sottostante $S(T) = S(\delta t n)$) sarà dato da:

$$S(T) = S(0)\, e^{\left(r-q-\frac{\sigma^2}{2}\right)T+\sigma\sqrt{T}\,\frac{1}{\sqrt{n}}\,\sum_{i=1}^{n} z_i^{(d)}} \;, \tag{2.30}$$

e in base al teorema del limite centrale: $\frac{1}{\sqrt{n}}\sum_{i=1}^{n} z_i^{(d)}$ converge ad una distribuzione gaussiana con media nulla e varianza unitaria, ovvero si recupera la variabile stocastica z e quindi la classica equazione nel continuo (2.28).
A questo punto, partendo dall'equazione (2.29), l'evoluzione dell'azione dopo un singolo step, può assumere unicamente due valori:

$$S(\delta t) = \begin{cases} S_0\, u & \text{movimento verso l'alto di S con probabilità } 1/2 \\ S_0\, d & \text{movimento verso il basso di S con probabilità } 1/2 \end{cases} \tag{2.31}$$

dove

$$\begin{aligned} u &= e^{\left(r-q-\frac{\sigma^2}{2}\right)\delta t+\sigma\sqrt{\delta t}} \qquad \text{(ottenuto imponendo } z^{(d)} = 1) \;, \\ d &= e^{\left(r-q-\frac{\sigma^2}{2}\right)\delta t-\sigma\sqrt{\delta t}} \qquad \text{(ottenuto imponendo } z^{(d)} = -1) \;, \end{aligned} \tag{2.32}$$

L'equazione riportata sopra consente di costruire algoritmicamente l'intero albero, reiterando più volte il processo sopra definito.
Rispetto al metodo CRR, la probabilità associata ai vari rami è sempre tenuta fissa al 50%, il prezzo da pagare per tale semplicità sta nel fatto che l'albero generato non risulta più simmetrico rispetto all'asse $S = S_0$ (vedi figura 2.4).

2.2.5 Estensione al caso di due sottostanti correlati

Rubinstein ha proposto un metodo abbastanza semplice per costruire un albero nel caso di due sottostanti correlati. Si tratterà quindi di un albero in tre dimensioni (una dimensione per ciascuno dei due sottostanti e la terza

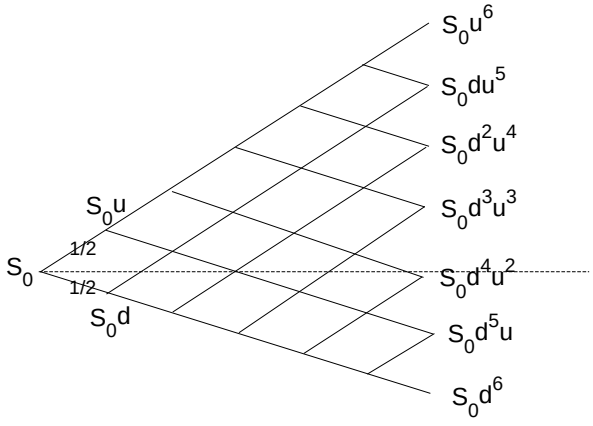

Figura 2.4: Albero binomiale di Rubinstein.

dimensione dedicata come al solito al tempo).
Si consideri quindi il generico nodo dell'albero (S_1, S_2), partendo da quest'ultimo avremo solo quattro possibilità, ciascuna con probabilità 0.25:

$$\begin{array}{ll} (S_1u_1, S_2u_2) & S_1 \text{ e } S_2 \text{ tendono a salire ,} \\ (S_1u_1, S_2d_2) & S_1 \text{ sale mentre } S_2 \text{ tenderebbe a decrescere ,} \\ (S_1d_1, S_2\hat{u}_2) & S_1 \text{ decresce e } S_2 \text{ tenderebbe a salire ,} \\ (S_1d_1, S_2\hat{d}_2) & S_1 \text{ e } S_2 \text{ decrescono .} \end{array} \tag{2.33}$$

Per ricavare i valori di u_1, d_1, u_2, d_2, $\hat{u}_2$, e $\hat{d}_2$, adottiamo il medesimo principio seguito nel caso di un albero bidimensinale, andiamo cioè a scrivere l'equazione stocastica per i due sottostanti S_1 e S_2 al tempo δt partento dai valori "iniziali" $S_1(0)$ e $S_2(0)$:

$$S_1(\delta t) = S_1(0)\, e^{\left(r-\frac{\sigma_1^2}{2}\right)\delta t+\sigma_1\sqrt{\delta t}\, w_1} , \tag{2.34}$$

$$\tag{2.35}$$

$$S_2(\delta t) = S_2(0)\, e^{\left(r-\frac{\sigma_2^2}{2}\right)\delta t+\sigma_2\sqrt{\delta t}\, w_2} , \tag{2.36}$$

dove w_1 e w_2 sono due variabili stocastiche correlate con correlazione pari a ρ. Possiamo quindi esprimere le due variabili w_1 e w_2 in termini di due variabili normali scorrelate z_1 e z_2 nel seguente modo (vedi paragrafo 2.1.4.d):

$$\begin{aligned} w_1 &= z_1 \\ w_2 &= \rho\, z_1 + \sqrt{1-\rho^2}\, z_2 . \end{aligned} \tag{2.37}$$

è infatti facile dimostrare che: $\langle w_1^2 \rangle = \langle w_2^2 \rangle = 1$ e $\langle w_1 \, w_2 \rangle = \rho$.
Ora utilizzando lo stesso argomento che abbiamo visto nel caso di un albero su di un singolo sottostante, sostituiamo z_1 e z_2 (variabili stocastiche normali) con altrettante variabili stocastiche bimodali che possono assumere solo i valori ± 1, con uguale probabilità (ovvero 50% in entrambi i casi). In tal modo la varianza e la media delle variabili z_1 e z_2 non cambiano e nel limite in cui si consideri un albero a più stadi molto fitto, si recuperera il limite continuo dato dalle equazioni (2.35)[5].
In tal modo possiamo ricavare la seguente valorizzazione per i parametri del nostro albero tridimensionale:

$$\begin{aligned} u_1 &= e^{\left(r-\frac{\sigma_1^2}{2}\right)\delta t+\sigma_1\sqrt{\delta t}} && (z_1 = 1) \, , \\ d_1 &= e^{\left(r-\frac{\sigma_1^2}{2}\right)\delta t-\sigma_1\sqrt{\delta t}} && (z_1 = -1) \, , \end{aligned} \tag{2.38}$$

e

$$\begin{aligned} u_2 &= e^{\left(r-\frac{\sigma_2^2}{2}\right)\delta t+\sigma_2\,(\rho+\sqrt{1-\rho^2})\,\sqrt{\delta t}} && (z_1 = 1 \text{ e } z_2 = 1) \, , \\ d_2 &= e^{\left(r-\frac{\sigma_2^2}{2}\right)\delta t+\sigma_2\,(\rho-\sqrt{1-\rho^2})\,\sqrt{\delta t}} && (z_1 = 1 \text{ e } z_2 = -1) \, , \\ \hat{u}_2 &= e^{\left(r-\frac{\sigma_2^2}{2}\right)\delta t-\sigma_2\,(\rho-\sqrt{1-\rho^2})\,\sqrt{\delta t}} && (z_1 = -1 \text{ e } z_2 = 1) \, , \\ \hat{d}_2 &= e^{\left(r-\frac{\sigma_2^2}{2}\right)\delta t-\sigma_2\,(\rho+\sqrt{1-\rho^2})\,\sqrt{\delta t}} && (z_1 = -1 \text{ e } z_2 = -1) \, , \end{aligned} \tag{2.39}$$

L'estensione del metodo definito sopra al caso di n sottostanti risulta relativamente immediato una volta che si consideri la decomposizione di Cholesky (la quale permette di esprimere, tramite una matrice triangolare, n variabili normali correlate in termini di n variabili normali scorrelate, ovvero l'esatto analogo del sistema di equazioni (2.37)).
In generale però un albero in alte dimensioni diventa computazionalmente inefficiente rispetto ad altri metodi numerici (come ad es. il Monte Carlo). La costruzione di alberi in alte dimensioni è quindi una divagazione di scarsa rilevanza pratica.

2.2.6 Alberi Binomiali a più stadi

Vediamo ora come sia possibile effettuare il pricing di un generico derivato tramite gli alberi binomiali. L'algoritmo è il seguente:

[5] Infatti per il teorema del limite centrale la somma di varibili identicamente distribuite converge ad una gaussiana

- Una volta stabilite le regole per la costruzione di un albero binomiale ad uno stadio (ovvero con un solo step di ampiezza δt), iterando nuovamente il procedimento è possibile costruire l'intero albero, con t che va da zero al tempo finale $T = n\,\delta t$. In tale costruzione ad ogni nodo viene assegnato un valore del sottostante e ad ogni ramo dell'albero (ovvero ogni segmento che connette due nodi) viene associata una probabilità.

- Dopo aver costruito l'albero si procedere con la seconda parte del lavoro che consiste nel costruire a ritroso il prezzo dell'opzione che si intende valutare, partendo dal pay-off finale ($t = T$).
  In pratica si procede algoritmicamente nel seguente modo:

  - Indicato con $t = T - i\delta t$ il tempo, si consideri inizialmente $i = 0$. Si riportano quindi sui nodi dell'albero corrispondenti al tempo a scadenza dell'opzione $t = T$, i valori del pay-off. Questo può essere fatto in quanto il valore dell'opzione a scadenza è noto per definizione.

  - Si esegue un ciclo iterativo a ritroso, determinando il valore dell'opzione, F, sui nodi al tempo $t = T - (i+1)\delta t$, conoscendo i valori al tempo $t = T - i\delta t$, tramite un semplice valore d'aspettazione:

$$F(i,j) = e^{-r\delta t}\left[p\,F(i+1,j+1) + (1-p)\,F(i+1,j-1)\right]\,, \tag{2.40}$$

  - Arrivati al tempo $t = 0$ (ovvero $i = n$), è possibile ottenere la stima del prezzo dell'opzione data da $F(n,0)$.

2.2.7 Alberi Binomiali: vantaggi e svantaggi

In generale la tecnica dell'albero binomiale ha i seguenti vantaggi:

- rispetto al metodo Monte Carlo, consente il pricing di opzioni con esercizio americano o bermudano;

- è estremamente semplice da implementare.

Ha però una serie di svantaggi:

- è meno robusto rispetto ad altri metodi simili, come il metodo alle differenze finite che presenteremo nel capitolo 2.3.

- Ha seri problemi all'aumentare della dimensionalità del problema, dove diventa rapidamente inefficiente rispetta ad esempio al metodo Monte Carlo.

- Più grave da un punto di vista strettamente operativo, l'albero binomiale risulta difficilemente estensibile ad opzioni con pay-off differenti, richiedendo di volta in volta, un'implementazione ad hoc. A parte le opzioni con barriera, dove l'estensione è immediata, non è banale vedere come applicare la tecnica dell'albero binomiale ad es. al pricing di opzioni asiatiche.
  Al contrario, sotto questo punto di vista, il metodo Monte Carlo risulta ottimale, in quanto la generazione dei cammini prescinde dal tipo di contratto, permettendo quindi di arrivare ad una gestione industriale del problema (ovvero ogni nuovo contratto richiede l'implementazione solo della nuova funzione di pay-off, mentre il motore di calcolo rimane sempre lo stesso). Questo aspetto viene generalmente trascurato dagli accademici ma è un punto essenziale che ha decretato forse più di ogni altro, l'enorme successo del Monte Carlo in finanza.

Da quanto detto sopra, emerge in maniera evidente un buco all'interno delle metodologie numeriche, precisamente quando il tipo di contratto che si vuole valutare presenta un esercizio di tipo bermudano o americano ed allo stesso tempo risulta scritto su più contratti o preveda diverse date di fixing (la qual cosa implica un'alta dimensionalità del problema). In tali casi infatti, il metodo Monte Carlo non è applicabile [6] e d'altra parte nemmeno la tecnica dell'albero binomiale o alle differenze finite, risultano fornire un'alternativa valida a causa dell'alta dimensionalità del problema. Non è quindi un caso che sul mercato strumenti di questo tipo (alta dimensionalità abbinato ad un esercizio di tipo non europeo) siano molto rari.

[6] In realtà in anni recenti sono state proposte delle estensioni che consentono di trattare in una certa misura l'esercizio anticipato anche all'interno del metodo Monte Carlo

2.3 Differenze Finite

In questo capitolo illustreremo il metodo esplitito alle differenze finite.

2.3.1 Introduzione e schemi di discretizzazione

Il metodo delle differenze finite, consiste nel andare a risolvere numericamente l'equazione di Black & Scholes (1.43) tramite una procedura di discretizzazione, ovvero trasformando le variabili continue del problema in variabili discrete e approssimando le derivate (tipicamente derivate parziali) in quozienti di differenze finite. Vedremo nei prossimi paragrafi, punto per punto, come implementare questo schema di discretizzazione per il problema formulato nel continuo.
In generale è utile osservare che nella matematica finanziara, come nella fisica, un utile modo di procedere, per risolvere un problema analiticamente (ovvero in forma chiusa), consiste nel formularlo inizialmente tramite un modello discreto per poi passare successivamente alla corrispondente versione continua. Infatti, usualmente, le equazioni nel continuo sono più facili da risolvere delle corrispondenti equazioni nel discreto. Ad es. Black & Scoles riuscirono a derivare una semplice formula chiusa in quanto il problema fu da loro formulato nel continuo (infatti assunsero che il prezzo di un'azione potesse variare con continuità mentre nella realtà i prezzi azionari sono soggetti a variazioni discrete). È però curioso osservare che nel momento in cui si cerca di risolvere lo stesso problema numericamente, la situazione tenda a rovesciarsi: la formulazione del problema nel discreto risulta generalmente più abordabile di quella nel continuo. La cosa in effetti non deve sorprendere eccessivamente in quanto i calcolatori operano sempre in termini di grandezze discrete e finite.

2.3.2 Schemi di discretizzazione

Vediamo quindi come sia possibile riformulare l'equazione di Black & Scholes nel discreto.

2.3.2.a Schema di discretizzazione per le variabili fondamentali

L'equazione che dobbiamo andare a risolvere numericamente è l'equazione di Black & Scholes (1.43), ovvero un'equazione alle derivate parziali in due variabili: S (prezzo dell'azione) e t (il tempo). Dovremo quindi adottare per tali variabili il seguente schema di discretizzazione:

$$S = j\,\delta S \qquad \text{con: } 0 \le j \le J\ , \tag{2.41}$$

$$t = i\,\delta t \qquad \text{con: } 0 \le i \le I \text{ e } \delta_t = T/I. \tag{2.42}$$

Per quanto riguarda la prima equazione, il valore $J\,\delta S$ rappresenta il valore massimo cha andremo a simulare all'interno del nostro schema numerico per il valore del sottostante. In teoria dovremmo quindi considerare valori di J molto alti, in pratica sarà sufficiente andare ad esplorare valori di $S_{\max} = J\,\delta S$ pari a tre o quattro volte il prezzo di esercizio di un opzione. Addiritttura nei casi un cui l'opzione abbia una barriera superiore B oltre il quale il pay off si annulla (ad es. in un'opzione up & out), si potrà prendere J tale che $S_{\max} = B$.

2.3.2.b Schema di discretizzazione delle funzioni

In generale una funzione $F(t, S)$ potrà essere rappresentata in maniera discreta dividendo lo spazio (t,S) in una griglia di punti secondo lo schema di discretizzazione specificato nelle equazioni (2.41) e (2.42), e rappresentando i valori della funzione sulla griglia come:

$$F_{i,j} = F(i\delta t, j\delta S) = F(t, S)\,, \tag{2.43}$$

ovvero tramite una matrice bidimensionale di dimensioni IxJ.

2.3.2.c Schema di discretizzazione per le derivate

Un punto fondamentale nel metodo delle differenze finite consiste nell'approssimare delle derivate (tipicamente derivate parziali) tramite dei quozienti di differenze finite. Il primo passo è quindi quello di trovare delle formule discrete che appossimino il meglio possibile delle derivate prime o seconde. A tal fine, consideriamo l'espansione in serie di Taylor per una generica funzione $g(x)$, arrestata al secondo ordine:

$$g(x+h) = g(x) + g'(x)\,h + \frac{1}{2}\,g''(x)\,h^2 + O(h^3)\,, \tag{2.44}$$

e consideriamo anche l'equazione gemella ottenuta scambiando h con $-h$:

$$g(x-h) = g(x) - g'(x)\,h + \frac{1}{2}\,g''(x)\,h^2 + O(h^3)\,, \tag{2.45}$$

è immediato ottenere (sommando in un caso e sottraendo nel secondo, ambo i membri delle equazioni (2.44) e (2.45)), le seguenti approssimazioni per la derivata prima, $g'(x)$ e $g''(x)$:

$$g'(x) = \frac{g(x+h) - g(x-h)}{2h} + O(h^2)\,, \tag{2.46}$$

$$g''(x) = \frac{g(x+h) - 2g(x) + g(x-h)}{h^2} + O(h^2)\,. \tag{2.47}$$

2.3.3 Discretizzazione dell'equazione di BS - metodo esplicito

Se riconsideriamo l'eq. di Black & Scholes (1.43) ed introduciamo le seguenti greche:

$$\theta = \frac{\partial F}{\partial t} \,, \tag{2.48}$$

$$\Delta = \frac{\partial F}{\partial S} \,, \tag{2.49}$$

$$\Gamma = \frac{\partial^2 F}{\partial S^2} \,, \tag{2.50}$$

possiamo riscrivere l'eq. (1.43) come:

$$\theta(S,t) + rS\Delta(S,t) + \frac{1}{2}\sigma^2 S^2 \Gamma - rF = 0 \,. \tag{2.51}$$

Se ora utilizziamo gli schemi di discretizzazione visti nel paragrafo precedente, ed approssimiamo le derivate Δ e Γ al tempo t con le corrispondenti derivate al tempo $t + \delta t$, possiamo scrivere:

$$\theta = \frac{F_{i+1,j} - F_{i,j}}{\delta_t} + O(\delta t) \,, \tag{2.52}$$

$$\Delta = \frac{F_{i+1,j+1} - F_{i+1,j-1}}{2\delta S} + O(\delta S^2) + O(\delta t) \,, \tag{2.53}$$

$$\Gamma = \frac{F_{i+1,j+1} - 2F_{i+1,j} + F_{i+1,j-1}}{\delta S^2} + O(\delta S^2) + O(\delta t) \,, \tag{2.54}$$

e quindi l'equazione di BS in formato discreto diventa:

$$\begin{aligned} &\frac{F_{i+1,j} - F_{i,j}}{\delta_t} + rS\,\frac{F_{i+1,j+1} - F_{i+1,j-1}}{2\delta S} + \\ &+\frac{1}{2}\sigma^2 S^2 \frac{F_{i+1,j+1} - 2F_{i+1,j} + F_{i+1,j-1}}{\delta S^2} - \\ &-rF_{i,j} = O(\delta t) + O(\delta S^2) \,. \end{aligned} \tag{2.55}$$

Riordinando i termini dell'equazione (2.56), otteniamo:

$$F_{i,j} = A_i\, F_{i+1,j-1} + B_i\, F_{i+1,j} + C_i\, F_{i+1,j+1} + O(\delta t^2) + O(\delta t \cdot \delta S^2) \,, \tag{2.56}$$

dove:

$$\begin{aligned} A_j &= \frac{\pi_{-1}}{1 + r\delta t} \,, \\ B_j &= \frac{\pi_0}{1 + r\delta t} \,, \\ C_j &= \frac{\pi_{+1}}{1 + r\delta t} \,. \end{aligned} \tag{2.57}$$

dove:

$$\begin{aligned} \pi_{-1} &= \frac{1}{2}\left(\sigma^2 j^2 - rj\right)\delta t\ , \\ \pi_0 &= 1 - \sigma^2 j^2 \delta t\ , \\ \pi_{+1} &= \frac{1}{2}\left(\sigma^2 j^2 + rj\right)\delta t\ . \end{aligned} \tag{2.58}$$

Una rappresentazione grafica della relazione (2.56) è data in figura 2.5.

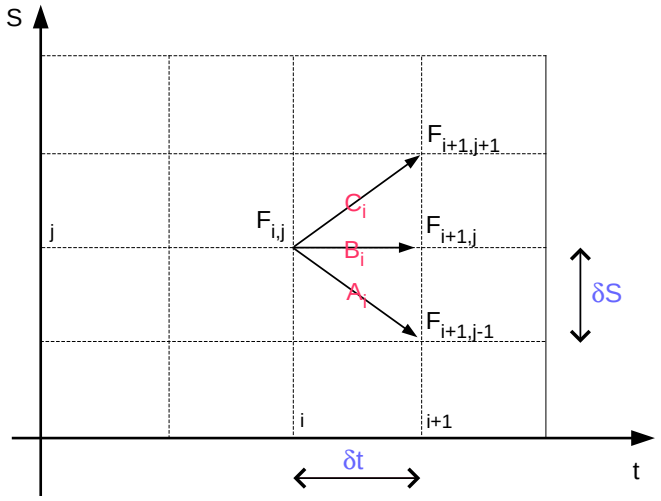

Figura 2.5: Metodo alle differenze finite: la figura mette in evidenza la relazione che lega F al tempo $t = i\,\delta t$ com i valori di F al tempo $t = (i+1)\,\delta t$.

2.3.3.a Condizioni ai bordi

Il set di equazioni (2.56) vale unicamente per i punti interni alla griglia (ovvero $1 <= j <= J-1$) e quindi, fissato i, si hanno solamente $J-1$ equazioni in $J+1$ incognite (in quanto $0 <= j <= J$). Per risolvere il problema è necessario quindi trovare altre due equazioni, queste verranno ottenute fissando delle opportune condizioni al contorno sui bordi della griglia, ovvero per $j = 0$ e $j = I$.

- Consideriamo il caso $S = 0$ (ovvero j=0), dall'equazione di Balck & Scholes (1.43) otteniamo per $S = 0$ la seguente equazione nel continuo:

$$\frac{\partial F(0,t)}{\partial t} - rF(0,t) = 0\ . \tag{2.59}$$

  quindi utilizzando lo schema di discretizzazione, otteniamo nel discreto l'equazione corrispondente:

$$F_{i,0} = B_0\, F_{i+1,0}\ , \tag{2.60}$$

la quale consente di trattare il problema sul bordo inferiore $j = 0$. In tal modo una volta noto il valore della funzione F sul bordo inferiore ($j = 0$) per il tempo $i + 1$ possiamo estrapolare il valore, sempre sul bordo, al tempo i, applicando l'equazione riportata sopra.

- Per $S = \infty$ (e quindi per $j = J$ con J molto grande), se ipotizziamo che il payoff dell'opzione sia lineare in S [7], abbiamo che: $\frac{\partial^2 F(S,t)}{\partial S^2} \to 0$ ovvero applicando la solita discretizzazione:

$$F_{i,J} = 2F_{i,J-1} - F_{i,J-2} \; . \tag{2.61}$$

  In tal modo una volta risolto il problema all'interno della griglia per un certo i, possiamo estrapolare il valore sul bordo ($j = J$) applicando l'equazione riportata sopra.

2.3.3.b Risoluzione dell'equazione discreta tramite un processo iterativo

Fissate quindi le condizioni ai bordi è possibile risolvere tramite una ricorsione di tipo "backward" (ovvero a ritroso) il set di equazioni (2.56), (2.60) e (2.61), ricavando $\{F_{i,j}\}$ partendo da $\{F_{i+1,j}\}$.
Come in tutti i processi ricorsivi, si dovrà partire da un punto iniziale predeterminato, nel nostro caso: $F_{i=I,j}$, il quale dipenderà strettamente dal tipo di contratto considerato, ovvero $F_{i=I,j} = \text{Pay–off}(j\delta S)$ [8]. La cosa rappresenta esattamente l'analogo di quanto abbiamo visto nel caso della risoluzione dell'equazione di Black & Scholes (vedi capitolo 1.3), in cui a fronte dell'equazione alle derivate parziali (1.43), il problema poteva poi essere risolto solo fissando opportune condizioni al contorno al tempo $t = T$ (dipendenti dal tipo di contratto considerato).
Il processo ricorsivo definito dal set di equazioni (2.56), (2.60) e (2.61), permette di calcolare, partendo dal dato noto $F_{I,j}$, il valore della funzione $F_{i,j}$ per ogni i, fino ad arrivare a $i = 0$ (ovvero $t = 0$). In tal modo otterremo il valore del derivato al tempo iniziale (cioè oggi) per ogni valore di j (ovvero per "ogni" possibile valore del sottostante al tempo iniziale): $F_{0,j} = F(t = 0, S = j\,\delta S)$. A questo punto risulta immediato ottenere il prezzo del derivato con le varie greche.
Si osservi che dovendo iterare il set di di equazioni (2.56), un numero di volte pari a $I = T/\delta t$, la stima del prezzo dell'opzione avrà una precisione dell'ordine di $O(\delta s^2)$ lungo la direzione "spaziale" e solo dell'ordine di $O(\delta t)$

[7] La qual cosa dipende dal tipo specifico di contratto considerato. Ad esempio nel caso di una call o di una put questa ipotesi è pienamente verificata. Diversamente sarà necessario ricercare l'opportuna condizione al contorno, analizzando il comportamento asintotico del prezzo dell'opzione quando il valore del sottostante divenga infinito.

[8] Ad esempio nel caso di un cotratto di tipo call avremo: $F_{i=I,j} = \text{Pay–off}(j\delta S) = \text{Max}\,(j\delta S - E, 0)$

lungo quella temporale.
Il metodo descritto sopra prende il nome di metodo alle differenze finite esplicito. Si usa l'espressione "differenze finite"a causa della discretizzazione operata. L'aggettivo esplicito si riferisce invece al fatto che il valore di $F_{i,j}$ ad ogni passo, si ottiene esplicitamente dai valori assunti da $F_{i+1,j}$ allo step immediatamente precedente.

2.3.4 Convergenza del metodo

Come tutti i metodi numerici, un fattore chiave anche nel caso del metodo alle differenze finite è la sua convergenza al valore "esatto". In effetti non sempre tale metodo converge, a meno di non fissare opportune condizioni sulla scelta dei parametri δt e δS (che nell'esposizione riportata sopra sono stati sempre lasciati liberi). È possibile dimostrare che una condizione necessaria e sufficiente per la convergenza dei risultati prodotti dal metodo alle differenze finite, consista nel scegliere δt e δS in modo tale da soddisfare le seguenti due relazioni:

$$\delta t \leq \frac{\delta S^2}{\sigma^2 S^2} \qquad \forall S \,, \tag{2.62}$$

$$\delta S \leq \frac{\sigma^2 S}{r} \qquad \forall S \,. \tag{2.63}$$

Senza entrare nel merito della dimostrazione di quanto asserito sopra, è interessante vedere come sia possibile derivare tali espressioni, esaminando la questione da un punto di vista finanziario. Il metodo alle differenze finite esplicito può essere visto come un albero trinomiale in quanto il valore del derivato, $F_{i,j}$, al tempo $i\,\delta t$, si ottiene come opportuna combinazione lineare del derivato al tempo, $(i+1)\,\delta t$, nei punti $j+1$, j e $j-1$. In tal caso i coefficienti π_{-1}, π_0 e π_{+1} vanno interpretate come le probabilità "risk neutral" dell'albero. E in effetti è immediato verificare che: $\pi_{-1}+\pi_0+\pi_{+1} = 1$, inoltre $E(S(t+\delta t) = -\delta S\,\pi_{-1} + 0\,\pi_0 + \delta S\,\pi_{+1} = r\,j\,\delta S\,\delta t = r\,S\,\delta t$ a conferma che l'incremento del valore dell'azione nell'intevallo di tempo δt avviene con il tasso privo di rischio. Affinche le "probabilità" π_{-1}, π_0 e π_{+1} possano però essere interpretate come delle vere probabilità di un albero trinomiale, è necessario imporre la condizione di positività, ovvero:

$$\sigma^2 j - r > = 0 \quad (\pi_{-1} \geq 0) \,, \tag{2.64}$$

$$1 - \sigma^2 j^2 \delta t > = 0 \quad (\pi_0 \geq 0) \,, \tag{2.65}$$

da cui si derivano le condizioni (2.62) e (2.63).

2.3.5 Esempi di applicazione del metodo delle differenze finite

2.3.5.a Call option con esercizio di tipo europeo

A titolo di esempio, riportiamo uno schema sintetico dell'algortimo da seguire per il calcolo del prezzo di un'opzione di tipo call europea, tramite il metodo esplicito alle differenze finite illustrato nei paragrafi precedenti.

- Si fissi il numero di step della griglia, Num_S_step, relativamente al prezzo dell'asset sottostante l'opzione e si determini quindi il passo della griglia associato alla variabile prezzo:
  $\delta S = \frac{2E}{\text{Num\_S\_step}}$. E è lo strike price dell'opzione.

- Si fissi il numero di step della griglia, relativamente alla dimensione temporale, Num_t_step e si determini il passo della griglia relativamente a tale direzione: $\delta t = T/\text{Num\_t\_step}$, dove T è la maturità dell'opzione. Al fine di garantire la convergenza, possiamo scegliere in prima battuta: $\delta t = \left(\frac{\delta S}{2\,E\,\sigma}\right)^2$, calcoliamo poi $\text{Num\_t\_step} = \text{Int}(T/\delta t) + 1$ ed infine ridefiniamo: $\delta t = T/\text{Num\_t\_step}$.

- Condizione finale: si fissi la condizione finale, ad es. nel caso di una call: $F(i,0) = \text{Max}(S-E,0)$, mentre per una put: $F(i,0) = \text{Max}(E-S,0)$

- Loop ricorsivo backward (il cuore della routine): si esegue un ciclo di tipo for sul numero di step temporali, partendo dal punto finale ($i = Num\_t\_step - 1$) fino ad arrivare a $i = 0$.
  Per ogni i:

  - si esegue un ciclo per $j = 1$ fino a $j = Num\_S\_step - 1$, andando prima a calcolare A_j, B_j e C_j e poi $F(i,j)$ in base alla relazione ricorsiva (2.56).
  - Si trova il valore del derivato ai bordi: $F(i, j = 0)$ e $F(i, j = \text{Num\_S\_step})$, imponendo le condizioni (2.60) e (2.61) rispettivamente.

- Una volta ottenuto il vettore $F(i = 0, j)$, si stima il prezzo dell'opzione tramite un'interpolazione lineare tra i valori della griglia più vicini al valore del prezzo spot[9] del sottostante, S_0, : $j_{\text{prox}} = \text{Int}(S_0/\delta S)$

$$\text{Prezzo Opt} = F(0, j_{\text{prox}}) + \frac{F(0, j_{\text{prox}}+1) - F(0, j_{\text{prox}})}{\delta S}(S_0 - j_{\text{prox}}\delta S)\ , \tag{2.66}$$

[9] Ovvero $t = 0$.

2.3.5.b Call option con esercizio di tipo americano

Inserire nello schema del paragrafo precedente il caso di esercizio di tipo americano è estremamente semplice, è infatti sufficiente confrontare ad ogni passo il valore del derivato ottenuto tramite la relazione (2.56) con il valore che si otterrebbe dall'esercizio anticipato. Il valore effettivo $F(i,j)$ è dato dal massimo tra queste due grandezze. La modifica al codice è quindi minimale.
Al fine di rendere più chiaro il modo in cui si trattano le opzioni americane tramite il metodo delle differenze finite, consideriamo un problema proveniente da un altro ambito. Questo sarà utile per evidenziare come da un lato certe metodologie non sono ristrette alla finanza, dall'altro come in realtà ci siano delle similitudini notevoli tra ambiti apparentemente diversi.

Esercitazione Si consideri il seguente gioco d'azzardo: un giocatore ha a disposizione n lanci di un dado. Ad ogni lancio può decidere di farsi pagare un quantitativo di denaro pari al numero del dado uscito nell'ultimo lancio moltiplicato per una posta fissa di 100 euro, oppure proseguire ritentando la sorte fino a quando non arriverà al numero massimo di lanci consentiti, pari a n. Si chiede qual'è il giusto prezzo da pagare all'inizio del gioco, per poter partecipare.

2.3.5.c Risultati numerici e metodi di accelerazione

In questo paragrafo riportiamo brevemente alcuni risultati numerici relativi al calcolo del prezzo di una call/put plain vanilla con esercizio di tipo europeo, confrontati con il valore esatto derivante dalla formula di Black & Scholes. Successivamente faremo vedere come l'uso di appropriate tecniche di accelerazione, consentano di migliorare considerevolmente le stime ottenute con il metodo alle differenze finite.
In figura 2.6 è riportato l'andamento dell'errore relativo tra la valutazione numerica ottenuta con le differenze finite ed il risultato esatto, nel caso di un'opzione call europea. Come si può vedere all'aumentare del numero di step lungo S (ovvero diminuendo δS), l'errore commesso tende a calare progressivamente. È anche interessante notare l'alternanza dell'errore rispetto alla parità o meno del numero di step N_s (nell'esempio in questione si ha una sottostima nel prezzo della call per valori di N_s pari e una sovrastima per valori dispari).
Tenendo presente il grado di approssimazione con cui abbiamo ricavato le equazioni ricorsive (2.56) e (2.56), è intuibile che l'errore di valutazione commesso applicando il metodo alle differenze finite sarà dell'ordine di $1\delta S^2 \sim N_s^2$, ovvero:

$$\epsilon_a(N_s) \approx P_{\mathrm{df}}(N_s) - P_{\mathrm{esatto}} \approx \frac{K(N_s)}{N_s^2} \tag{2.67}$$

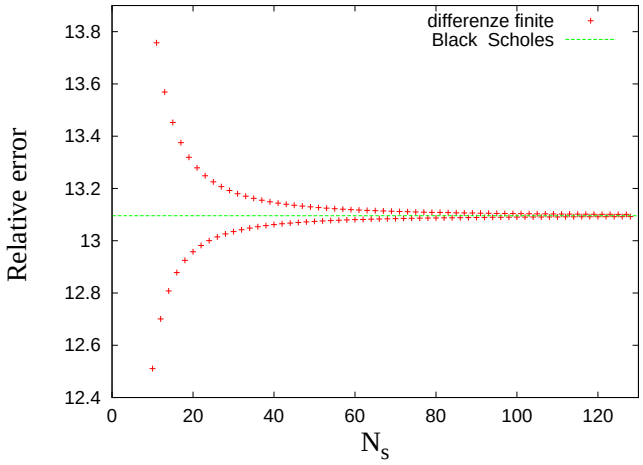

Figura 2.6: Stima del prezzo, con il metodo alle differenze finite, di un'opzione call europea con con i seguenti parametri: $\sigma = 20$ %, $S_0 = 100$, $r = 2$ %, $E = 100$ e $T = 2$ anni. La retta orizzontale rappresenta il valore esatto ottenuto con la formula di Black & Scholes.

dove:

$$K(N_s) = \begin{cases} K_P\ , & \text{se } N_s \text{ è pari} \\ K_D\ , & \text{se } N_s \text{ è dispari} \end{cases} \tag{2.68}$$

la dipendenza dalla parità di N_s, nel termine $K(N_s)$, è stata inserita al fine di tener conto dell'andamento alternato osservato nel grafico 2.6.
La verifica della relazione (2.67), è immediata una volta che si costruisca un grafico, in coordinate log-log, dell'errore relativo $\epsilon_r = P_{\text{df}}(N_s)/P_{\text{esatto}} - 1$ al variare di N_s (vedi figura 2.7).

Come si può constatare il fit con una retta con coefficiente angolare molto vicino a -2 risulta ottimo, confermando così quanto prescritto dalla relazione (2.67). È interessante osservare come la costante di proporzionalità K dipenda in generale dalla parità di N_s (ovvero sia diversa a seconda che si considerino valori interi pari o dispari per N_s). In particolare in questo caso, il valore assoluto di K_P è più piccolo di K_D, e ciò significa che le simulazioni con N_s pari consentono di raggiungere una maggiore precisione di quelle con N_s dispari (in altri termini la convergenza risulta più efficiente sul ramo dei valori pari).
La conoscenza dell'andamento dell'errore commesso tramite l'applicazione "cruda" del metodo alle differenze finite, può essere sfruttata al fine di definire delle procedure di accelerazione, le quali consentano di ottenere stime con una maggiore precisione a parità di sforzo numerico/computazionale. A tal fine, supponiamo di considerare due differenti valori di N_s: N_{s1} e N_{s2},

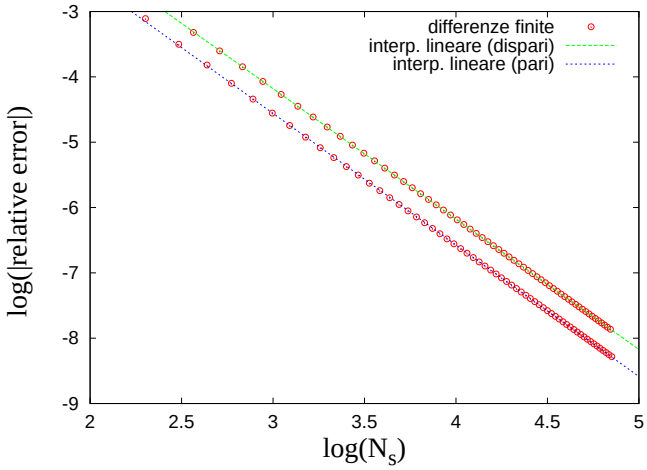

Figura 2.7: Errore relativo commesso con il metodo alle differenze finite per un'opzione call europea con con i seguenti parametri: $\sigma = 20$ %, $S_0 = 100$, $r = 2$ %, $E = 100$ e $T = 2$ anni. Il grafico è in formato log-log: in ascissa figura il logaritmo del numero di step spaziali, $\log(N_s)$, mentre in ordinata compare il logaritmo dell'errore relativo $\log(|\epsilon_r|)$. Le rette interpolanti, applicando il metodo dei minimi quadrati, sono: $1.48232 - 2.01383\log(N_s)$ per valori di N_s pari e $1.80828 - 1.99602\log(N_s)$ per valori di N_s dispari. Da cui si ottengono le seguenti stime: $K_P = -57.662$ e $K_D = 79.883$.

entrambi pari o dispari, possiamo allora scrivere le due equazioni:

$$\begin{aligned} \epsilon_a(N_{s1}) &= P_{\rm df}(N_{s1}) - P_{\rm esatto} \approx \frac{K}{N_{s1}^2} \\ \epsilon_a(N_{s2}) &= P_{\rm df}(N_{s2}) - P_{\rm esatto} \approx \frac{K}{N_{s2}^2} \ , \end{aligned}$$

da cui si ricava facilmente che:

$$P_{\rm esatto} \approx P_{\rm df}(N_{s1}) - \frac{P_{\rm df}(N_{s1}) - P_{\rm df}(N_{s2})}{1 - \left(\frac{N_{s1}}{N_{s2}}\right)^2} \ . \tag{2.69}$$

È immediato ottenere anche una stima di K:

$$K \approx \frac{P_{\rm df}(N_{s1}) - P_{\rm df}(N_{s2})}{\frac{1}{N_{s1}^2} - \frac{1}{N_{s2}^2}} \ . \tag{2.70}$$

A titolo esemplificativo, abbiamo effettuato la stima del prezzo della call, applicando il metodo d'accelerazione descritto nell'equazione (2.69), con i dati riportati in tabella 2.2.

La valutazione finale (riportata sempre in tabella 2.2) presenta un errore relativo dello 0.0045 %, di gran lunga inferiore alla semplice stima ottenuta

Metodo d'accelerazione per l'algoritmo alle differenze finite		
Stima prezzo di una call (differenze finite)	N_s	ϵ_r
$P_{\mathrm{df}}(20) = 12.9578$	20	-1.0529 %
$P_{\mathrm{df}}(40) = 13.0616$	40	-0.2598 %
$P_{\mathrm{df}}(\infty) \approx 13.0962$	∞	0.0045 %

Tabella 2.2: Opzione call con i seguenti parametri: $S(t_0) = 100$, $r = 2$ %, $T = 2$ anni e $E = 100$. L'ultima riga riporta la stima ottenuta applicando il metodo d'accelerazione all'algoritmo alle differenze finite. Il valore di K stimato in base ai parametri riportati in tabella risulta pari a -55.3885, il segno meno indica che considerando valori di N_s pari, il prezzo della call risulta sottostimato rispetto al valore esatto.

con il metodo alle differenze finite ad es. nel caso in cui $N_s = 128$ (dove l'errore relativo è pari a -0.0253 %).
È anche interessante valutare quale sarebbe stato lo sforzo computazionale necessario per raggiungere lo stesso livello di precisione ricorrendo all'algoritmo base. A tal riguardo ricordiamo che, nell'algoritmo alle differenze finite, il tempo di CPU scala proporzionalmente al numero di punti nella griglia, ovvero $N_s \cdot N_t$ e d'altra parte $N_t \sim 1/\delta t \sim 1/\delta S^2 \sim N_s^2$; quindi il tempo di calcolo scala complessivamente come N_s^3. Nel caso dell'esempio ripotato sopra, abbiamo perciò che il tempo di calcolo necessario ad ottenere la stima migliorata (dato dalla somma dei tempi per le due simulazioni con $N_s = 20$ e $N_s = 40$) è pari a 9 CPU time($N_s = 20$) [10]. A questo punto ricorrendo sempre alla formula (2.67), si può stimare che per avere un errore relativo pari a 0.0045 %, siano necessari circa 306 step spaziali [11], con un CPU time totale pari a 3585 CPU time($N_s = 20$), ovvero 398 volte il tempo di calcolo della stima migliorata. In altri termini a parità di precisione nella stima del prezzo, il metodo alle differenze finite abbinato con un sistema di accelerazione della convergenza, risulta quasi 400 volte superiore all'algorit-

[10] Prendendo come riferimento il tempo di calcolo per $N_s = 20$, abbiamo che: CPU time($N_s = 40$) ≈ 8 CPU time($N_s = 20$), da cui il risultato.

[11] Tale valore si ottiene applicando la formula:

$$N_s = \mathrm{Int}_{\{\mathrm{pari}\}} \left[\sqrt{\frac{K_P}{P_{\mathrm{esatto}} \cdot \epsilon_r}} \right] , \tag{2.71}$$

facilmente derivabile dalla (2.67), dove abbiamo supposto di considerare un numero di step spaziali pari ed abbiamo utilizzato per K_P la stima ottenuta in tabella 2.2. Se avessimo considerato valori di N_s dispari, il numero di step necessario a raggiungere la precisione desiderata sarebbe stato uguale a circa 369.

mo base! Un risultato sorprendente.
Analogamente a quanto abbiamo visto nel caso del Monte Carlo, dove tipicamente si utilizzano metodi di riduzione della varianza al fine di migliorare la qualità della stima a parità di tempo macchina, anche nel caso delle differenze finite è quindi importante cercare di migliorare l'algoritmo base con opportune tecniche di accelerazione.

2.3.6 Vantaggi e punti deboli del metodo alle differenze finite

Rispetto all'albero binomiale (di cui è stretto parente in base a quanto abbiamo visto) il metodo alle differenze finite risulta molto più robusto e solido. È quindi da preferirsi sempre all'albero binomiale. Soffre però degli stessi limiti che abbiamo già visto a proposito dell'albero binomiale nel paragrafo 2.2.7, rispetto alla questione della dimensionalità e della scarsa estendibilità a pay–off generici.

Capitolo 3

Progetto di una libreria in C++ per il “pricing”

3.1 Progetto per una libreria finanziaria in C++

3.1.1 Introduzione

In questa sezione definiremo sinteticamente le linee guida per costruire una semplice libreria finanziaria. La libreria verrà scritta in C++, quindi ogni oggetto finanziario (ad es. curva dei tassi, bond, opzioni ecc.) verrà rappresentato tramite un'opportuna classe.

3.1.2 Obiettivi della libreria

L'obiettivo che ci poniamo è quello di creare una libreria finanziaria che ci consenta di: 1) gestire le curve dei tassi; 2) effettuare il pricing dei bond; 3) descrivere le opzioni su equity; 4) effettuare il pricing delle opzioni su equity tramite formule chiuse (esatte o approssimate) e metodi numerici (Monte Carlo, alberi binomiali e metodi alle differenze finite).

Il primo passo, per definire una libreria di classi, consiste nell'analizzare il problema, isolando i principali oggetti coinvolti. Nel presente caso, possiamo individuare le seguenti macro aree:

(A) Curva dei tassi (paragrafo 3.1.4).

(B) Bond (paragrafo 3.1.5).

(C) Pricing bond (paragrafo 3.1.6).

(D) Azioni (paragrafo 3.1.7).

(E) Processi stocastici (paragrafo 3.1.8).

(F) Opzioni (paragrafo 3.1.9).

(G) Pricing di opzioni su equity (Monte Carlo, Albero Binomiale, Differenze Finite e Formule chiuse) (paragrafo 3.1.10).

Naturalmente, sarà necessario disporre di una libreria con funzionalità matematiche già sviluppate, da utilizzarsi nella costruzione della nostra libreria finanziaria. A tal riguardo, un'ottima scelta è la libreria freeware GNU Scientific Library sviluppata dalla GNU foundation.[1], in alternativa si può far riferimento agli algoritmi di *Numerical Recipes* [23].

[1]Il codice sorgente della libreria GNU Scientific Library è disponibile al sito: http://www.gnu.org/software/gsl. Inoltre esistono versioni già compilate per l'ambiente windows, in particolare presso il sito: http://sourceforge.net/project/showfiles.php?group_id=23617 alla voce gsl, dove sono reperibili anche i sorgenti e la documentazione sull'utilizzo delle routine della libreria.

3.1.3 Avvertenze ed utilizzo delle presenti note

Le note di seguito allegate, costituiscono semplicemente una traccia per lo sviluppo della libreria. Il loro obiettivo è quello di definire con chiarezza: a) la struttura, ovvero lo scheletro della libreria e b) le principali funzioni con cui i vari oggetti interagiscono tra di loro; non è quello di definire nei dettagli ogni singolo oggetto o di elencare in maniera esaustiva tutte le funzioni membro delle varie classi. Sarà compito degli studenti completare ciascun oggetto (ovvero rifinire nei dettagli la struttura delle classi) ed implementare le funzioni proprie di ciascuna classe.

La notazione utilizzata per la descrizione grafica delle classi e delle loro relazioni segue lo standard UML. Richiamiamo brevemente di seguito il significato delle notazioni più importanti di questo standard.

- Una classe viene rappresentata tramite un rettangolo diviso in tre sezioni. Nella prima (quella che si trova nella parte alta), figura il nome della classe; nella seconda sezione vengono riportati gli attributi ed infine nell'ultima parte compaiono i metodi. Ad esempio in figura 3.1 viene riportata una classe per raccogliere i dati anagrafici di una persona. Il nome della classe è Persona, gli attributi sono: il nome, il cognome e la data di nascità. Infine figura un metodo, Get_age, il quale consente di calcolare l'età della persona partendo dalla sua data di nascita e dalla data attuale.

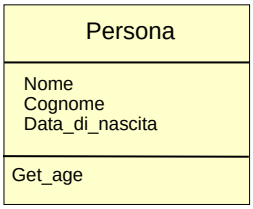

Figura 3.1: Esempio notazione UML.

- La relazione tra due classi (detta associazione), viene descritta tramite una linea che le unisce. In generale esiste un'associazione tra due istanze di due classi se l'istanza di una classe deve conoscere l'altra

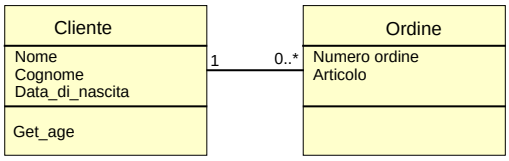

Figura 3.2: Esempio notazione UML.

al fine di effettuare correttamente il proprio lavoro. Nell'esempio di fig. 3.2, è stata rappresentata la relazione tra la classe Cliente e la classe Ordine. I numeri ai lati della linea che rappresenta l'associazione, indicano la molteplicità, ovvero il numero di possibili istanze della classe, associate con una singola instanza dell'altra. Nel nostro esempio per ogni cliente ci possono essere più ordini, mentre per un dato ordine ci può essere un solo cliente. Così la molteplicità della classe Cliente è 1, mentro la molteplicità della classe Ordine è 0..* intendendo con questo che può essere un numero qualunque a partire da zero.
La relazione tra due classi può comprendere delle frecce di navigabilità indicanti la direzionalità dell'associazione. Ad esempio in fig. 3.3 la

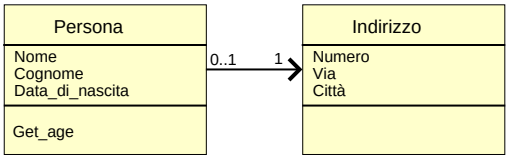

Figura 3.3: Esempio notazione UML.

freccia sta ad indicare che una istanza della classe Persona "possiede" un indirizzo (o più propriamente un'istanza della classe Indirizzo) e non viceversa. È importante notare che graficamente la freccia non è piena (la freccia piena verrà utilizzata per indicare la relazione di ereditarietà).

- Nel caso in cui una classe sia derivata da un'altra (tramite ereditarietà), si utilizzera una freccia piena per indicare la relazione che le lega. La freccia piena punterà verso la classe padre partendo dalla

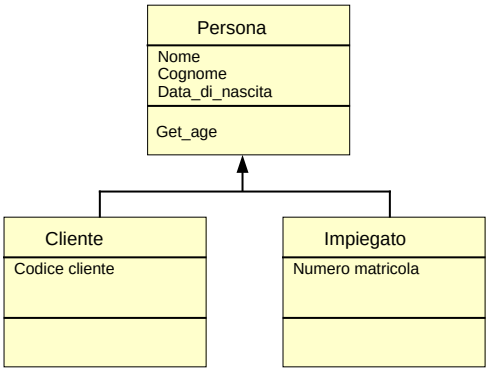

Figura 3.4: Esempio notazione UML.

classe figlio. Un esempio è riportato in figura 3.4, dove viene mostrato

il rapporto gerarchico che lega le classi “Cliente” e “Impiegato” alla classe padre “Persona”. Ovviamente un cliente ed un impiegato sono casi particolari di una categoria più ampia, quella delle persone.

Nei paragrafi successivi vengono definite le varie classi, raggruppate in base alle macroaree definite nel capitolo 3.1.2.

3.1.4 Curva dei tassi

- Descrizione: questa sezione è dedicata alla gestione delle curve sui tassi di interesse.
- Il diagramma di questo settore della libreria e' riportato nella figura sottostante. Le frecce indicano la derivazione della classe sottostante partendo da quella sovrastante (ovvero le due classi Yield_curve_flat e Yield_curve_term_structure sono entrambe derivate dalla classe astratta padre, Yield_curve).

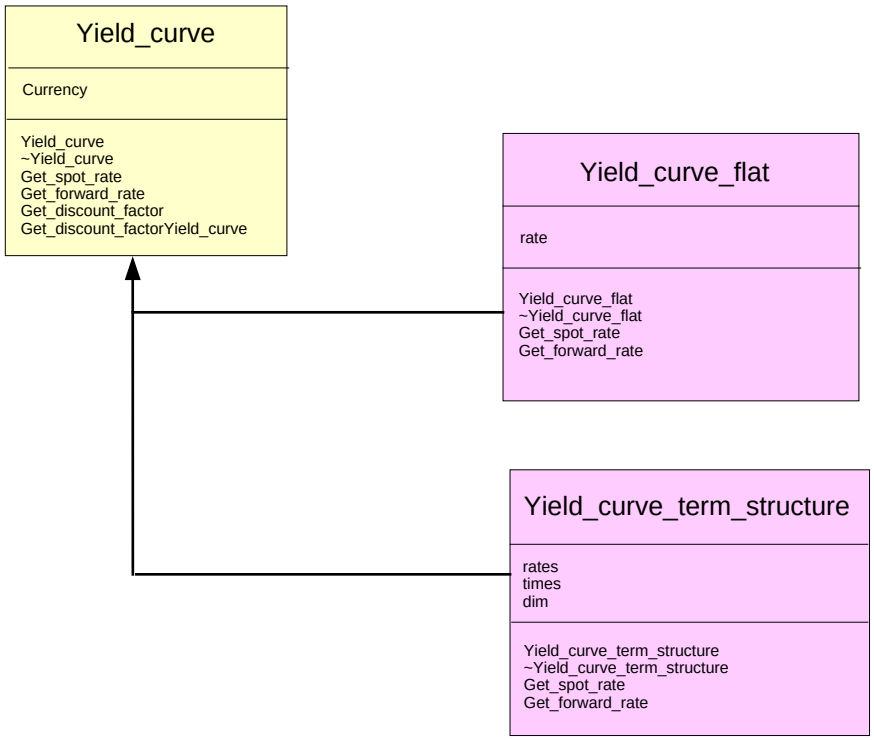

Figura 3.5: Schema curve tassi.

Gli scheletri per le classi sopra riportate sono schematizzate nelle pagine seguenti.

3.1.4.a Classe: Yield_curve

- Nome classe: Yield_curve
- Significato: Classe astratta per gestire la curva dei tassi.

Scheletro della classe (file .hpp)

```
// -------------------- Begin yield_curve.hpp ----------------------

class Yield_curve {

  private:
      char *currency ;

  public:

      // Constructors
      //
      Yield_curve(void) ;

      Yield_curve(char *currency_init) ;

      // Destructor
      //
      virtual ~Yield_curve(void) ;

      // Functions
      //
      virtual double Get_spot_rate(double t) = 0 ;

      virtual double Get_forward_rate(double t_start,
                                      double t_end) = 0 ;

      double Get_discount_factor(double t) ;

      double Get_discount_factor(double t_start,
                                 double t_end) ;

} ;

// -------------------- End yield_curve.hpp ------------------------
```

Significato dei membri della classe

(1) currency: è la valuta di riferimento per la curva dei tassi in oggetto.

(2) Get_spot_rate(double t): restituisce lo spot rate al tempo t, ovvero il tasso di interesse tra oggi e la data t (rappresentata come year fraction). Il tasso è espresso in capitalizzazione continua.
La funzione è dichiarata come virtuale pura.

(3) Get_forward_rate(double t_start, double t_end): restituisce il tasso forward calcolato sulla curva tra il tempo t_start e t_end. I tempi sono espressi come year fraction. Il tasso è rappresentato in capitalizzazione continua.
La funzione è dichiarata come virtuale pura.

(4) Get_discount_factor(double t): restituisce il fattore di sconto calcolato tra oggi e la data t (epressa come year fraction).

(5) Get_discount_factor(double t_start, double t_end): restituisce il fattore di sconto calcolato tra t_start e la data t_end (epresse entrambe come year fraction).

Esercizio

Si completi eventualmente la classe Yield_curve riportata sopra e si implementino le funzioni membro di tale classe in un opportuno file "yield_curve.cpp".

3.1.4.b Classe: Yield_curve_flat

- Nome classe: Yield_curve_flat
- Significato: Classe derivata da Yield_curve per gestire una curva dei tassi piatta, ovvero senza alcuna term structure. Questa è il tipo più semplice di curva che si possa ipotizzare.

Scheletro della classe (file .hpp)

```
// -------------------- Begin yield_curve_flat.hpp -----------------

class Yield_curve_flat {

  private:
      double rate ;

  public:

      // Constructors
      //
      Yield_curve_flat(void) ;

      Yield_curve_flat(char *currency_init, double rate_init) ;


      // Functions
      //
      virtual double Get_spot_rate(double t) ;

      virtual double Get_forward_rate(double t_start, double t_end) ;

} ;

// -------------------- End yield_curve_flat.hpp -------------------
```

Significato dei membri della classe

(1) rate: tasso di interesse in capitalizzazione continua.

(2) Yield_curve_flat(char *currency_init, double rate_init): costruttore della classe Yield_curve_flat. Riceve in input il nome della currency a cui la curva dei tassi si riferisce ed il tasso flat espresso in capitalizzazione continua.

Esercizio

Si implementino in un file "yield_curve_flat.cpp" tutte le funzioni membro della classe : Yield_curve_flat.

3.1.4.c Classe: Yield_curve_term_structure

- Nome classe: Yield_curve_term_structure
- Significato: Classe derivata da Yield_curve per gestire una curva dei tassi con term structure definita.

Scheletro della classe (file .hpp)

```
// ------------- Begin yield_curve_term_structure.hpp --------------

class Yield_curve_term_structure {
  private:
        double *rates ;
        double *times ;
        int dim ;
        .
        .
  public:

        // Constructors
        //
        Yield_curve_term_structure(void) ;

        Yield_curve_term_structure(char *currency_init,
                                   double *rate_deposit,
                                   double *t_deposit,
                                   int num_depositi,
                                   double *rate_swap,
                                   double *t_swap,
                                   int num_swap) ;

        Yield_curve_term_structure(char *currency_init,
                                   double *rates_init,
                                   double *times_init,
                                   int dim_init) ;
        // Destructor
        //
        virtual ~Yield_curve_term_structure(void) ;

        // Functions
        //
        virtual double Get_spot_rate(double t) ;

        virtual double Get_forward_rate(double t_start, double t_end) ;
} ;

// ------------- End yield_curve_term_structure.hpp ----------------
```

Significato dei membri della classe

(1) rates è un vettore di tassi e times è un vettore di tempi (espressi in termini di year fraction) avente uguale dimensione dim. rates[i] rappresenta il tasso risk free in capitalizzazione continua al tempo times[i].

(2) Yield_curve_term_structure(double *rete_deposit, ...): costruttore della classe Yield_curve_term_structure. Riceve in input un vettore di tassi deposito, relativi alle scadenze specificate dal vettore t_depositi, di lunghezza num_depositi, e un vettore di tassi swap, relativi alle scadenze specificate dal vettore t_swap, di lunghezza num_swap.
Tramite un algoritmo di bootstrapping viene costruita la struttura a termine della curva dei tassi, associata ai tessi deposito e swap passati al costruttore.

(3) Yield_curve_term_structure(double *rates_init, ...): costruttore della classe Yield_curve_term_structure. Riceve in input un vettore di tassi rates_init e corrispondentemente un vettore di tempi (espressi in year fraction) avente uguale dimensione dim_init. rates_init[i] rappresenta il tasso risk free in capitalizzazione continua al tempo times_init[i].

3.1.5 Anagrafica bond

- Descrizione: questa sezione è dedicata alla definizione di una classe per gestire l'anagrafica delle obbligazioni (bond), sia a tasso fisso che a tasso variabile.
- Il diagramma di questo settore della libreria e' riportato nella figura sottostante.

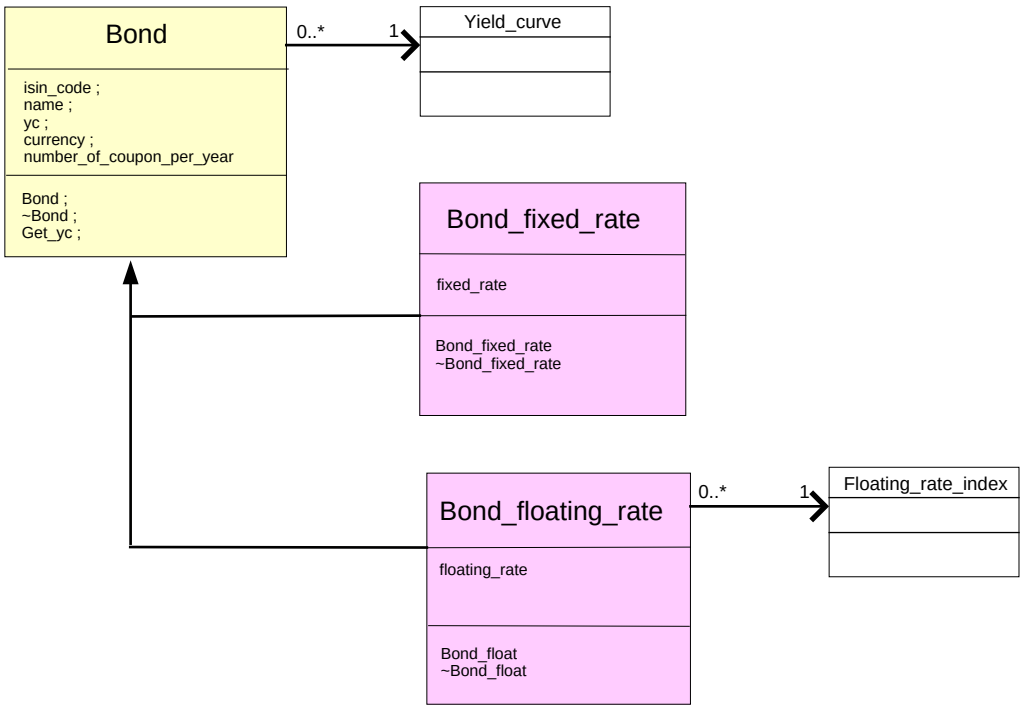

Figura 3.6: Schema anagrafica bond.

3.1.5.a Classe: Bond

- Nome classe: Bond
- Significato: Classe per gestire l'anagrafica di un bond.

Scheletro della classe (file .hpp)

```
// -------------------- Begin bond.hpp -----------------------------

class Bond {

  private:
      char *isin_code ;
      char *name ;
      Yield_curve *yc ;
      char *currency ;
      int number_of_coupon_per_year ;
      .
      .
      .

  public:

      // Default constructor
      //
      Bond(void) ;

      // Destructor
      //
      virtual ~Bond(void) ;

      Yield_curve *Get_yc(void) {
           return yc ;
      } ;

      .
      .
      .

} ;

// -------------------- End bond.hpp -------------------------------
```

Significato dei membri della classe

(1) isin_code: stringa contenente l'isin code del titolo.

(2) name: nome per esteso del bond.

(3) yc: puntatore ad un oggetto di tipo Yield_curve che rappresenta la curva dei tassi con cui scontare i flussi di cassa del titolo.

(4) currency: stringa contenente la valuta dei flussi di pagamento del titolo.

(5) number_of_coupon_per_year: numero di coupon staccati per anno.

(6) Get_yc: restituisce il puntatore all'oggetto yc.

Esercizio

Si completi la classe definita sopra con tutti gli altri campi anagrafici essenziali per una descrizione esaustiva di un titolo obbligazionario.

3.1.5.b Classe: Bond_fixed_rate

- Nome classe: Bond
- Significato: Classe per gestire l'anagrafica di un bond.

Scheletro della classe (file .hpp)

```
// -------------------- Begin bond_fixed_rate.hpp ------------------

class Bond_fixed_rate : public Bond {

  private:
      double fixed_rate ;

  public:

      // Default constructor
      //
      Bond_fixed_rate(void) {
      };

      // Destructor
      //
      virtual ~Bond_fixed_rate(void) ;

} ;

// -------------------- End bond_fixed_rate.hpp --------------------
```

Significato dei membri della classe

(1) fixed_rate: tasso di interesse su base annua.

3.1.5.c Classe: Bond_float

- Nome classe: Bond_float
- Significato: Classe per gestire l'anagrafica di un bond indicizzato ad un tasso variabile.

Scheletro della classe (file .hpp)

```
// -------------------- Begin bond_floating_rate.hpp ---------------

class Bond_floating_rate : public Bond {

  private:
      Floating_rate_index *floating_rate ;

      // TO BE COMPLITED

  public:

      // Default constructor
      //
      Bond_float(void) ;

      // Destructor
      //
      virtual ~Bond_float(void) ;

      // TO BE COMPLITED

} ;

// -------------------- End bond_floating_rate.hpp -----------------
```

Significato dei membri della classe

(1) floating_rate: indice di tipo Floating_rate_index a cui il bond è indicizzato.

Esercizio

Si completi la definizione della classe riportata sopra per la gestione dei un bond indicizzato ad un tasso variabile e si proceda all'implementazione delle relative funzioni. A tal fine sarà necessario strutturare nei dettagli la classe Floating_rate_index (il cui fine è quello di caratterizzare e descrivere un indice floating).

3.1.6 Metodi per il pricing dei bond

- Descrizione: questa sezione è dedicata alla definizione di una classe per gestire il pricing delle obbligazioni (bond).
- Il diagramma di questo settore della libreria e' riportato nella figura sottostante.

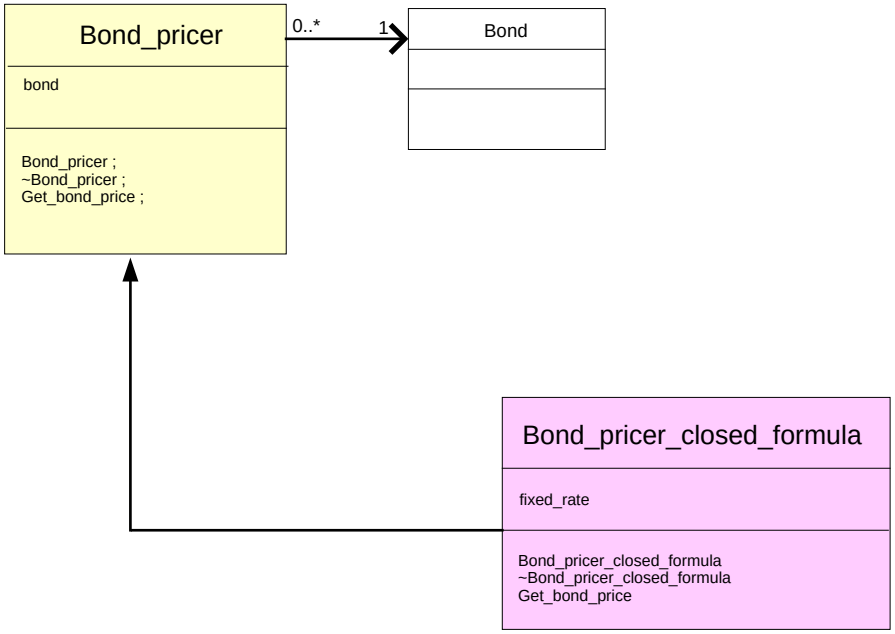

Figura 3.7: Schema bond pricer.

3.1.6.a Classe: Bond_pricer

- Nome classe: Bond_pricer
- Significato: Classe astratta da cui verranno derivate le altre classi.

Scheletro della classe (file .hpp)

```
// -------------------- Begin bond_pricer.hpp ----------------------

class Bond_pricer {

  private:
      Bond *bond ;

      // TO BE COMPLITED

  public:
      // Constructors & destructors

      // Default constructor
      Bond_pricer(void) ;

      // Constructor
      Bond_pricer(Bond *bond_init) {
           bond = bond_init ;
      };

      // Destructor
      virtual ~Bond_pricer(void) ;


      virtual double Get_bond_price(void) = 0 ;


} ;

// -------------------- End bond_pricer.hpp ------------------------
```

Significato dei membri della classe

(1) bond: puntatore ad un oggetto di tipo Bond. bond rappresenta l'obbligazione di cui si vule determinare il prezzo tramite il metodo in oggetto.

(2) Bond_pricer(Bond *bond_init): è il costruttore per la classe Bond_pricer.

(3) Get_bond_price(void): funzione virtuale pura che restituisce il valore (prezzo) del titolo. Tale funzione andrà opportunamente implementata nelle classi derivate da Bond_pricer.

3.1.6.b Classe: Bond_pricer_exact_closed_formula

- Nome classe: Bond_pricer_exact_closed_formula
- Significato: Classe per gestire il pricing tramite formula chiusa.

Scheletro della classe (file .hpp)

```
// ---------------- Begin bond_pricer_closed_formula.hpp ------------

class Bond_pricer_closed_formula  : public Pricer_bond {

  private:
      // TO BE COMPLITED

  public:
      // Constructors & destructors

      // Default constructor
      Bond_pricer_closed_formula(void):  Bond_pricer() {
      };

      virtual double Get_bond_price(void) ;

      // TO BE COMPLITED

} ;

// ----------------End bond_pricer_closed_formula.hpp --------------
```

Significato dei membri della classe

(1) Get_bond_price(void): in questa classe la funzione per il pricing del bond andrà implementata utilizzando il noto algoritmo basato sul discounting dei flussi di cassa futuri.

Esercizio

Si implementino le funzioni membro della classe Bond_pricer_exact_closed_formula, con particolare riguardo per la funzione Get_bond_price.

3.1.7 Anagrafica e prezzi di azioni

- Descrizione: L'insieme delle classi di questa sezione ha come finalità nel suo complesso, quella di permettere la descrizione e la gestione delle azioni (a livello anagrafico) e dei relativi prezzi (quotazioni). Tra le classi presenti in questa sezione, figura anche la "Volatility_surface" il cui fine è quello di descrivere la superficie di volatilità associata ad un'azione. Nella classe deputata a rappresentare l'anagrafica di un'azione, figureráppunto un pointer ad un oggetto di tipo Volatility_surface.
  Questa sezione sarà molto importante per quanto riguarda la costruzione di una libreria di pricing, in quanto le azioni sono i tipici sottostanti dei contratti opzionali.

- Il diagramma di questo settore della libreria e' riportato nella figura sottostante.

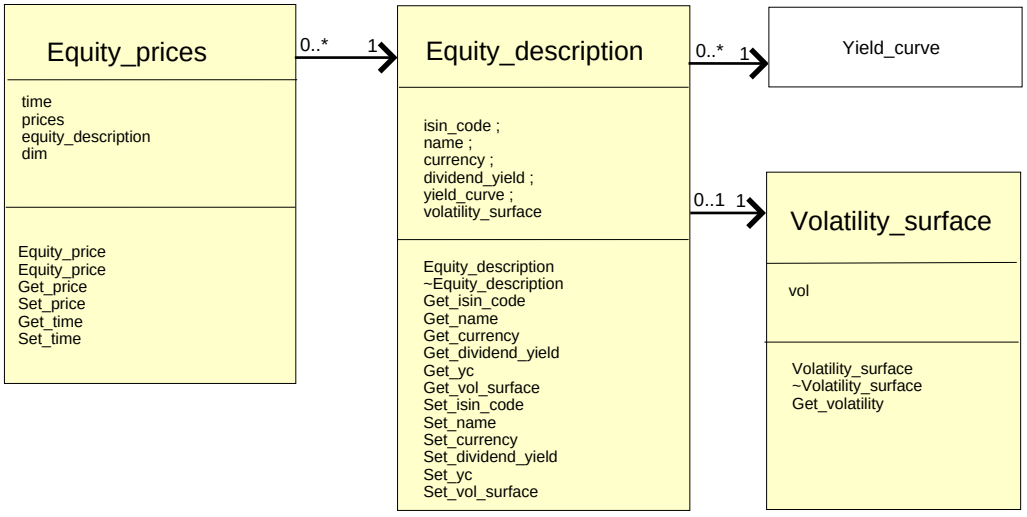

Figura 3.8: Schema anagrafica e prezzi azioni

3.1.7.a Classe: Volatility_surfaces

- Nome classe: Volatility_surfaces
- Significato: Classe per gestire le volatilità di un sottostante azionario. Si parla di superficie di volatilità in quanto si può avere, in generale, una dipendenza di questa grandezza da due fattori: (i) da un lato vi può essere una vera e propria struttura a termine per la volatilita', relativamente all'asse temporale (analogamente a quanto accade per la curva dei tassi). Ovvero la volatilità dell'azione tra l'istante attuale ed il tempo t potrà dipendere da t; (ii) vi è anche una dipendenza della volatilità dal valore dell'azione (a causa degli effetti di smile, si veda a tal proposito il paragrafo 1.3.6).
  Nella classe riportata sotto abbiamo considerato solo il caso estremamente banale in cui non vi sia dipendenza temporale o dipendenza dal valore del sottostante; è facile però creare un'opportuna gerarchia di classi di superfici di volatilità al fine di gestire situazioni più complesse (sulla falsa riga di quanto fatto precedentemente per le curve dei tassi).

Scheletro della classe (file .hpp)

```
// ------------------ Begin volatility_surface.hpp -----------------

class Volatility_surface {

  private:
      double vol ;

  public:

      // Default constructor
      //
      Volatility_surface(void) ;

     double Get_volatility(double t_start, double t_end) {
          return vol ;
     } ;

} ;

// ------------------ End volatility_surface.hpp -------------------
```

Significato dei membri della classe

(1) vol: è la volatilità dell'azione.

(2) Get_volatility: restituisce la volatilità dell'azione nell'intervallo di tempo compreso tra t_start e t_end. Se t_start non corrisponde all'istante attuale ma è una data nel futuro, la funzione restituisce una volatilità forward. Nel caso in questione, poiché la superficie di volatilità si riduce ad un punto, la funzione Get_volatility si limita a restituire sempre la costante vol.

3.1.7.b Classe: Equity_description

- Nome classe: Equity_description
- Significato: Classe per gestire l'anagrafica di una singola azione. Ogni azione è caratterizzata da una serie di campi (detti campi anagrafici) che la descrivono completamente. Ad esempio il nome dell'azione, il suo codice isin [2], la valuta in cui sono espresse le quotazioni, il mercato principale su cui il titolo viene trattato, i dividendi pagati con le relative date ecc. ecc. La classe riportata di seguito ha lo scopo di raccogliere tutti questi campi in un unico costrutto che costituirà appunto l'anagrafica dell'azione.

Scheletro della classe (file .hpp)

```
// ------------------- Begin equity_description.hpp -----------------

class Equity_description {

  private:
      char *isin_code ;
      char *name ;
      char *currency ;
      double dividend_yield ;
      Yield_curve *yield_curve ;
      Volatility_surface *volatility_surface ;

  public:

      // Default constructor
      //
      Equity_description(void) ;

      // Destructor
      //
      virtual ~Equity_description(void) ;

      Volatility_surface *Get_vol_surface(void) {
          return volatility ;
      } ;

     Yield_curve *Get_yc(void) {
          return yc ;
     } ;
```

[2] Il codice isin è una stringa alfanumerica che identifica in maniera univoca un'azione o un titolo obbligazionario.

```
     double Get_dividend_yield(void) {
          return dividend_yield ;
     } ;

     // TO BE COMPLITED
} ;

// ------------------ End equity_description.hpp -------------------
```

Significato dei membri della classe

(1) isin_code: isin code dell'azione

(2) name: nome per esteso dell'azione

(3) currency: valuta con cui si esprime il prezzo dell'azione in oggetto.

(4) yc: puntatore ad un oggetto di tipo Yield_curve, rappresentante la curva dei tassi associata alla currency dell'azione.

(5) dividend_yield: dividend yield su base annua che caratterizza il titolo azionario.

(5) volatility_surface: curva di volatilità dell'azione (si veda per maggiori dettagli la descrizione della classe Volatility_curve).

Esercizio

Si completi la definizione della classe con gli eventuali campi anagrafici mancanti. Si arricchisca la classe con tutte le funzioni di interfaccia necessarie e con le relative implementazioni.

3.1.7.c Classe: Equity_prices

- Nome classe: Equity_prices
- Significato: Classe per gestire i prezzi del/dei sottostanti. Questa classe definisce l'involucro dove immagazzinare i prezzi di una o più azioni. In generale un modo molto semplice di definire questa classe è quella di inserire nella sua struttura: un membro double per rappresentare l'istante temporale a cui il prezzo o i prezzi si riferiscono ed un secondo membro di tipo double *, dove archiviare i prezzi delle azioni. Questa semplice scelta non consente di apprezzare a fondo il vantaggio di definire una classe dedicata per rappresentare i prezzi azionari, va infatti osservato che ad un dato istante temporale, per una singola azione, sono presenti sul mercato pi'ǒprezzi, ad esempio il prezzo bid, l'ask [3] e il mid (media tra i primi due). Vi sono poi a livello giornaliero i prezzi di apertura, quelli di chiusura ecc. È quindi utile pensare di definire una classe che consenta di trattare tutti questi tipi di prezzo tramite un'unica struttura. Nell'implementazione riporata sotto ci limiteremo comunque a considerare il semplice caso in cui vi sia un'unica tipologia di prezzi (possiamo pensare a questi valori come ai prezzi mid di un insieme di azioni).

Scheletro della classe (file .hpp)

```
// ------------------ Begin equity_prices.hpp ----------------------

class Equity_prices {

  private:
      double time ;
      double *prices ;
      Equity_description **equity_description ;
      int dim ;

  public:

      // Default constructor
      //
      Equity_prices(void) ;

      Equity_prices(double time_init, double *prices_init,
```

[3] Si parla di prezzo bid (detto anche prezzo denaro) relativamente allacquisto da parte di un market maker che quota un certo titolo e di un prezzo ask (detto anche prezzo lettera) per la vendita. È quindi intuibile che il prezzo bid sarà sempre più basso di quello ask.

```
                Equity_description **equity_description_init,
                int dim) ;

    double Get_time(void) {
        return time ;
    } ;

    double Set_time(double time_init) {
        time = time_init ;
    } ;

    double Get_price(int i) {
        if(i>=0 && i<dim) {
                return prices[i] ;
        }
        else {
            // Error
            exit(1) ;
        }
    } ;

    int Set_price(int i, double price_init) {
        if(i>=0 && i<dim) {
                prices[i] = price_init ;
        }
        else {
            // Error
            exit(1) ;
        }
    } ;

} ;


// ------------------ End equity_prices.hpp ------------------------
```

Significato dei membri della classe

(1) dim: numero di azioni di cui la classe definisce i prezzi.

(2) prices: è un vettore (di lunghezza dim) contenente i prezzi (mid) delle "dim" azioni in oggetto, al tempo specificato da time.

(3) time: tempo espresso in year fraction a cui il vettore di prezzi prices si riferisce.

(4) equity_description: pointer a pointer di tipo Equity_description. È in pratica un vettore di lunghezza dim in cui il generico elemento i-esimo

equity_description[i] rappresenta un pointer ad un oggetto di tipo Equity_description che contiene l'anagrafica dell'azione il cui prezzo al tempo "time" è dato da prices[i].

(5) Equity_prices(double time_init, ...): costruttore della classe. Consente di inizializzare i membri di tipo dato che caratterizzano la classe.

(6) Get_price(int i): restituisce il prezzo dell'azione i-esima.

(7) Set_price(int i, double price_init): imposta il prezzo dell'azione i-esima al valore specificato da price_init.

Si osservi che se si volesse estendere la classe riportata sopra al fine di trattare oltre al prezzo mid (rappresentato dal vettore double *prices) anche i prezzi bid e ask, sarebbe sufficiente introdurre nella classe dei vettori ulteriori:

```
double *price_bid ;
double *price_ask ;
```

imponendo ad es. che:

```
prices[i] = 0.5 * ( price_bid[i] + price_ask[i] ) ;
```

in alternativa (con notevoli vantaggi per la coerenza dei dati) si può addirittura sopprimere in questo caso il vettore double *prices, lasciando solo i due vettori price_bid e price_ask e modificando la funzione Get_price(int i) in modo che fornisca "on demand" il prezzo dell'azione i-esima come media di price_bid[i] e price_ask[i].

Esercizio

Si completi la classe con tutte le funzioni di interfaccia necessarie.

3.1.8 Processi stocastici e generazione di cammini

- Descrizione: questa sezione è dedicata ai processi stocastici, con particolare riguardo a quelli utilizzati in finanza per modellare le azioni. Come già emerso nella descrizione dei processi stocastici fatta nel capitolo 1.2, una dei tratti fondamentali che li caratterizza è la presenza di un fattore d'aleatorietà, per cui partendo da un dato prezzo azionario al tempo t, il valore futuro dell'azione al tempo $t + \delta t$ non è stabilito in maniera deterministica ma può tipicamente assumere infiniti valori, ciascuno con una propria densità di probabilità. In defnitiva, dato un generico processo stocastico, sarà necessario estrarre una o più variabili da un generatore di numeri casuali (con distribuzione opportuna) al fine di calcolare il punto finale d'arrivo (ad es. il prezzo dell'azione al tempo $t+\delta t$). Per questa ragione sarà quindi opportuno inserire tra i membri di una eventuale classe per gestire i processi stocastici, un generatore di numeri pseudo-casuali con distribuzione uniforme (e/o gaussiana).
  La gerarchia riportata in fig. 3.9, per la gestione dei processe stocastici, parte da una classe astratta padre (Process) che si divide in due flussi: Process_eq per i processi stocastici riguardanti le azioni e Process_IR per descrivere l'evoluzione aleatoria dei tassi di interesse. Nel presente volume approfondiremo unicamente le classi associate a processi stocastici azionari. Non verrà quindi definita in magggior dettaglio la parte riguardante i processi stocastici su tasso [4]. Infine sempre in questa macrosezione della libreria finanziaria, vengono introdotte tre ulteriori classi: Random_numbers, Schedule e Path (vedi figura 3.10), in quanto strettamente collegate alla gerarchia delle classi riguardanti i processi stocastici. In particolare Random_numbers è una semplice classe per trattare un vettore di double, la sua finalità sarà quella di archiviare i random numbers necessari ad un generico processo stocastico per ottenere un futuro valore del (o dei) sottostanti a partire da un punto dato (il prezzo o i prezzi dell'azione oggi). Ad esempio nel caso di un processo stocastico log-normale per una singola azione, Random_numbers conterrà un vettore di lunghezza pari a 1, è infatti sufficiente un'unica variabile casuale estratta da una distribuzione gaussiana per generare un nuovo punto a partire da un punto dato (cioè il valore iniziale dell'azione). Nel caso invece di un processo stocastico lognormale multivariato per n azioni, Random_numbers dovrà ospitare un vettore di double di lunghezza pari a n.
  La classe Path permette di descrivere un cammino (spezzato) seguito da uno o più sottostanti azionari. Rappresenta cioè un possibile scenario evolutivo per le azioni sottostanti ad un contratto opzionale.

[4] Per una discussione approfondita su questo tema rimandiamo all'ottimo testo di Brigo e Mercurio [24].

Infine la classe Schedule consente di rappresentare un insieme di date (tipicamente le date di fixing di un contratto opzionale) e risulta importante in uno dei costruttori di Path.

- Il diagramma di questo settore della libreria e' riportato nelle figure 3.9 e 3.10.

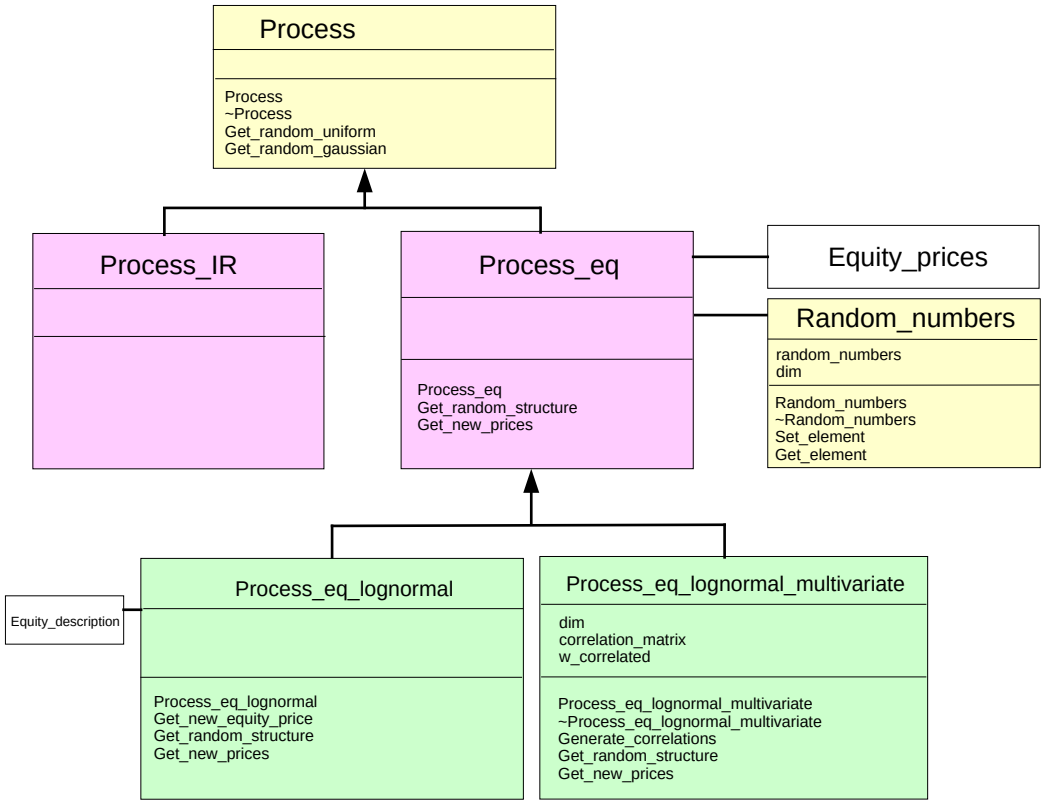

Figura 3.9: Schema generale per la gerarchia di classi riguardanti i processi stocastici.

3.1.8.a Classe: Process

- Nome classe: Process
- Significato: Classe astratta per gestire i processi stocastici

Scheletro della classe (file .hpp)

```
// ------------------ Begin process.hpp ---------------------------

class Process {

  private:
  .
  .

  public:

      // Default constructor
      //
      Process(void) {
      };

      // Destructor
      //
      virtual ~Process(void) {
      };

      // Random number generators
      //
      double Get_random_uniform(void) ;
      double Get_random_gaussian(vodi) ;
} ;

// ------------------ End process.hpp -----------------------------
```

(1) Get_random_uniform: funzione che restituisce un numero casuale distribuito uniformemente tra [0,1] ;

(1) Get_random_gaussian: funzione che restituisce un numero casuale distribuito normalmente con media zero e varianza unitaria.

Relativamente alle funzioni per la generazione di numeri casuali (con distribuzione uniforme e/o gaussiana), degli ottimi algortimi sono presenti nella libreria matematica GSL.

3.1.8.b Classe: Process_eq

- Nome classe: Process_eq
- Significato: Classe astratta per gestire i processi per le azioni

Scheletro della classe (file .hpp)

```
// ------------------ Begin process_eq.hpp ------------------------


class Process_eq {

  private:
  .
  .

  public:
      // Constructors & destructors

      // Default constructor
      //
      Process_eq(void) {
      };

      // Destructor
      //
      virtual ~Process_eq(void) {
      };


     virtual Random_numbers *w Get_random_structure(void) = 0 ;

     virtual Equity_prices *Get_new_prices(Equity_prices *eq_prices_in,
                                           Random_numbers *w,
                                           double delta_t) = 0 ;
} ;


// ------------------ End process_eq.hpp -------------------------
```

3.1.8.c Classe: Process_eq_lognormal

- Nome classe: Process_eq_lognormal
- Significato: Classe per descrivere un processo lognormale per una singola azione. L'elemento fondamentale di questa classe è la funzione Get_new_equity_price che restituisce il valore assunto dall'azione ad un istante futuro t_end, quando in input siano forniti: (a) l'anagrafica dell'azione Equity *equity_description (da cui si ricava il dividend yield, la volatilità e la curva dei tassi associata all'azione); (b) il valore iniziale dell'azione (equity_price), all'istante t_start; (c) la variabile aleatoria w; (d) il tempo iniziale t_start e (e) il tempo finale t_end.
  Più precisamente tale funzione implementa la formula (1.36) con l'unica accortezza di sostituire μ con il tasso risk free (essendo la valutazione dei derivati effettuata in un mondo neutrale al rischio).

Scheletro della classe (file .hpp)

```
// ------------------ Begin process_eq_lognormal.hpp ---------------


class Process_eq_lognormal {

  public:

     // Default constructor
     //
     Process_eq_lognormal(void) {
     };

     static double Get_new_equity_price(
                    Equity_description *equity_description,
                    double equity_price,
                    double w,
                    double t_start,
                    double t_end
                    ) ;

     virtual Random_numbers *w Get_random_structure(void) ;

     virtual Equity_prices *Get_new_prices(Equity_prices *eq_prices_in,
                                           Random_numbers *w,
                                           double delta_t) ;
} ;
```

```
// ------------------ End process_eq_lognormal.hpp -----------------
```

Significato dei membri della classe

(1) Get_new_equity_price: e' la funzione che restituisce il nuovo prezzo dell'azione (descritta dall'anagrafica equity_description) al tempo t_end, partendo dal prezzo dell'equity equity_price al tempo t_start. La variabile stocastica w dovrà essere passata in input.

Implementazione di alcune funzioni della classe (file .cpp)

```
// ------------------ Begin process_eq_lognormal.cpp ---------------


double Process_eq_lognormal::Get_new_equity_price(
                                Equity *equity_description,
                                double equity_price,
                                double w,
                                double t_start,
                                double t_end
                                ) {

    Yield_curve *yc = equity_description->Get_yc() ;
    double r = yc->Get_forward_rate(t_start, t_end) ;

    double dividend_yield = equity_description->Get_dividend_yield() ;

    double sigma = equity_description->Get_vol() ;

    double delta_t = t_end - t_start ;

    return equity_price * exp( (r - dividend_yield - 0.5*sigma*sigma) *
           delta_t + sigma * sqrt(delta_t) * w ) ;
}



Random_numbers *Process_eq_lognormal::Get_random_structure(void) {
    Random_numbers *w = new Random_numbers(1) ;
    w->Set_element(0, Get_random_gaussian() ) ;
    return w ;
}



Equity_prices *Process_eq_lognormal::Get_new_prices(
```

```
                                        Equity_prices *eq_prices_in,
                                        Random_numbers *w,
                                        double delta_t) {

    Equity_prices *eq_prices_out ;

    eq_prices_out = new Equity_prices(1) ;

    new_equity_price = Get_new_equity_price(
                         eq_prices_in->Get_equity_description(0),
                         eq_prices_in->Get_equity_price(0),
                         w->Get_element(0),
                         eq_prices_in->Get_time(),
                         eq_prices_in->Get_time() + delta_t) ;

    eq_prices_out->Set_equity_prices(0, new_equity_price) ;

    eq_prices_out->Set_time(eq_prices_in->Get_time()
                         + delta_t) ;

    eq_prices_out->Set_equity_description(0,
                         eq_prices_in->Get_equity_description(0)) ;

    return eq_prices_out ;
}


// ------------------- End process_eq_lognormal.cpp -----------------
```

3.1.8.d Classe: Process_eq_lognormal_multivariate

- Nome classe: Process_eq_lognormal_multivariate
- Significato: Classe per descrivere un processo lognormale per un set di azioni.

Scheletro della classe (file .hpp)

```
// ---------- Begin process_eq_lognormal_multivariate.hpp ---------


class Process_eq_lognormal_multivariate {

  private:
     int dim ;
     double **correlation_matrix ;
     Random_numbers *w_correlated ;
     .
     .
     .

  public:

     // Default constructor
     //
     Process_eq_lognormal_multivariate(void) ;

     Process_eq_lognormal_multivariate(int dim_init) ;

     // Destructor
     //
     virtual ~Process_eq_lognormal_multivariate(void) ;

     int Generate_correlations(Random_numbers *w_out,
                               Random_numbers *w_in) ;

     virtual Random_numbers *Get_random_structure(void) ;

     virtual Equity_prices *Get_new_prices(Equity_prices *in,
                                           Random_numbers *w,
                                           double delta_t) ;

} ;


// ---------- End process_eq_lognormal_multivariate.hpp -----------
```

Significato dei membri della classe

(1) Get_new_equity_price: e' la funzione che rerstiutisce il nuovo prezzo dell'azione (descritta dall'anagrafica equity_description) al tempo t_end, partendo dal prezzo dell'equity equity_price al tempo t_start.

Implementazione di alcune funzioni della classe (file .cpp)

```
// ------------- Begin process_eq_lognormal_multivariate.cpp -------

Random_numbers *Process_eq_lognormal_multivariate::Get_random_structure(
                            void) {

    Random_numbers *w = Random_numbers(dim) ;
    for(int i=0; i<dim; i++) {
         w->Set_element(i, Get_random_gaussian() ) ;
    }
    return w ;
}

Equity_prices *Process_eq_lognormal_multivariate::Get_new_prices(
                            Equity_prices *eq_prices_in,
                            Random_numbers *w,
                            double delta_t) {
  int i ;
  Equity_prices *eq_prices_out ;

  eq_prices_out = new Equity_prices(dim) ;

  // Introduce correlations
  //
  Generate_correlations(w_correlated, w) ;

  for(i=0; i<dim; i++) {

     new_equity_price = Process_eq_lognormal::Get_new_equity_price(
                           eq_prices_in->Get_equity_description(i),
                           eq_prices_in->Get_equity_price(i),
                           w_correlated->Get_element(i),
                           eq_prices_in->Get_time(),
                           eq_prices_in->Get_time()+ delta_t) ;

     eq_prices_out->Set_equity_prices(i, new_equity_price) ;

     eq_prices_out->Set_time(eq_prices_in->Get_time()
                                          + delta_t) ;
```

```
    eq_prices_out->Set_equity_description(i,
                    eq_prices_in->Get_equity_description(i)) ;

  }

  return eq_prices_out ;
};

// ------------- End process_eq_lognormal_multivariate.cpp ---------
```

Esercizio

Si completi la classe in oggetto, implementando le funzioni mancanti. Particolare cura dovrà essere posta nella gestione della matrice di varianza covarianza (si dovranno in particolare scrivere le funzioni di interfaccia necessarie e si dovrà prevedere nel costruttore della classe, l'opportuna inizializzazione della matrice correlation_matrix) e nella definizione della funzione Generate_correlations il cui compito consiste nel creare una n-upla di variabili correlate normali partendo da una n-upla di variabili scorrelate normali (si dovrà implementare a questo scopo l'algoritmo di decomposizione di Cholesky, si veda a tal proposito l'ottimo libro di Jackel [21]).

3.1.8.e Classe: Process_IR

- Nome classe: Process_IR
- Significato: Classe astratta per gestire i processi per i tassi

Scheletro della classe (file .hpp)

```
// ------------------ Begin process_IR.hpp ------------------------


class Process_IR {

  private:
  .
  .

  public:
      // Constructors & destructors

      // Default constructor
      //
      Process_IR(void) {
      };

      // Destructor
      //
      virtual ~Process_IR(void) {
      };


} ;


// ------------------ End process_IR.hpp --------------------------
```

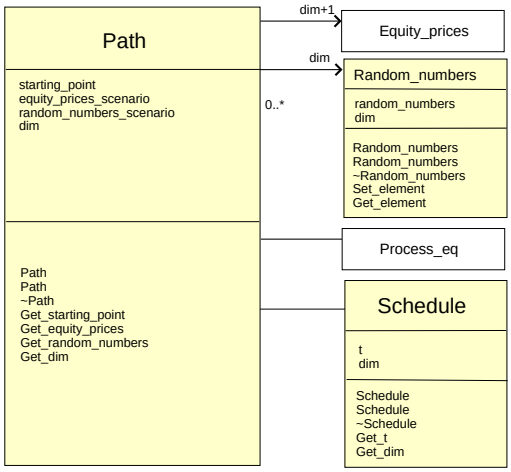

Figura 3.10: Schema in cui viene rappresentata la classe path e la sua relazione con altre classi delle libreria.

3.1.8.f Classe: Random_numbers

- Nome classe: Random_numbers
- Significato: Classe per gestire i numeri random necessari in un generico processo stocastico.

Scheletro della classe (file .hpp)

```
// ------------------ Begin random_numbers.hpp ---------------------


class Random_numbers {

  private:
      double *random_numbers ;
      int dim ;

  public:
      // Constructors & destructors

      // Default constructor
      //
      Random_numbers(void) ;

      Random_numbers(int dim) ;
```

```
      // Destructor
      //
      virtual ~Random_numbers(void) ;

      int Set_element(int i, double value) ;

      double Get_element(int i) ;

} ;


// ------------------- End random_numbers.hpp -----------------------
```

3.1.8.g Classe: Schedule

- Nome classe: Schedule
- Significato: Questa classe ha l'obiettivo di rappresentare un insieme di date ordinate cronologicamente. La classe conterrà quindi come elemento base un vettore di double di lunghezza dim. Ogni elemento del vettore rappresenterà una data futura espressa come intervallo di tempo (in unità di year fraction) rispetto ad una data di riferimento arbitraria presa come punto zero (ad es. il 1 gennaio 2000).
  A titolo esemplificativo, supponiamo di voler rappresentare tramite un oggetto della classe Schedule, le seguenti tre date: 1 luglio 2005, 1 gennaio 2006 e 1 gennaio 2007. Si supponga di utilizzare come data di riferimento dell'asse dei tempi il 1 gennaio 2000, l'oggetto Schedule dovrà contenere i setuenti tre numeri:

$$5.5 = \frac{\text{1 luglio 2005}-\text{1 gennaio 2000}}{365}$$
$$6.0 = \frac{\text{1 gennaio 2006}-\text{1 gennaio 2000}}{365}$$
$$7.0 = \frac{\text{1 gennaio 2007}-\text{1 gennaio 2000}}{365}$$

  Un elemento importante che dovrà caratterizzare la classe sarà il fatto che l'ordinamento cronologico andrà sempre garantito. Questo significa che gli eventuali costruttori di questa classe dovranno sempre verificare l'ordinamento crescente degli istanti temporali considerati.

Scheletro della classe (file .hpp)

```
// ------------------ Begin schedule.hpp ---------------------------

class Schedule {

  private:
      double *t ;
      int dim ;

  public:
      // Constructors & destructors

      // Default constructor
      Schedule(void) ;

      Schedule(double t_ref, double delta_t, int dim_init) ;

      Schedule(double *t_init, int dim_init) ;
```

```
      // Destructor
      virtual ~Schedule(void) ;


      // Funzioni di interfaccia
      //
      double *Get_t(int i) ;

      int Get_dim(vodi) ;

} ;


// ------------------ End schedule.hpp ----------------------------
```

Significato dei membri della classe

(1) t: è un pointer ad un vettore di tipo double, contenente una successione di tempi (in year fraction) ordinati cronologicamente.

(2) dim: è la dimensione del vettore t.

(3) Schedule(double t_ref, double delta_t, int dim_init): è uno dei costruttore della classe. Permette di descrivere un set di punti temporali equispaziati in cui l'intervallo di tempo che separa due punti successivi è pari a δt, ovvero: $t[i] = t_{\text{ref}} + (i+1) \cdot \delta t$, con $i = 0, 1....$dim-1. I punti t[i] sono generati a partire da un punto iniziale t_ref.

(4) Schedule(double *t_init, int dim_init): è uno dei costruttore della classe. Permette di inizializzare il vettore t a partire dal vettore t_init passato dall'esterno.

(5) Get_t(int i): restituisce le coordinate temporali del punto i-esimo espresse in termini di year fraction.

(6) Get_dim(void): restituisce il numero complessivo di date archiviate nell'oggetto Schedule (ovvero la dimensione del vettore t).

3.1.8.h Classe: Path

- Nome classe: Path
- Significato: Legata strettamente alla classe Process_eq (e nel seguito alla classe Option_pricer_monte_carlo), vi è la classe deputata a rappresentare i cammini di una o più azioni (tipicamente generati da una simulazione Monte Carlo). La classe in questione, Path, è definita dallo scheletro riportato sotto.
  In generale un cammino seguito da una o più azioni è completamente definito nel momento in cui si specifica:
  - il punto di partenza, il quale rappresenta il/i prezzi delle azioni al tempo iniziale. Nello scheletro della classe, questo è rappresentato dal pointer starting_point che punta ad un oggetto di tipo Equity_prices.
  - i valori assunti dalla/dalle azioni in una serie discreta di tempi futuri (il cui numero totale è rappresentato nella classe dalla variabile intera dim). Nello scheletro della classe questi punti, che rappresentano un possibile scenario evolutivo, sono identificati da equity_prices_scenario, un pointer a pointer di tipo Equity_prices. L'interpretazione di questa struttura è ben rappresentata in figura 3.11.

  In aggiunta nella classe è presente un pointer a pointer ad oggetti di tipo Random_numbers, il cui scopo è quello di raccogliere i dim oggetti di tipo Random_numbers utilizzati dal processo stocastico per la generazione del cammino. Ad esempio nel caso di un processo lognormale per una singola azione, random_numbers_scenario[0] è un pointer ad un oggetto di tipo Random_numbers contenente un'unica variabile di tipo double, la quale rappresenta la componente stocastica w utilizzata nella formula (1.36) per ottenere il prezzo dell'azione $S(t_1)$ al tempo t_1 partendo dal valore dell'azione $S(t_0)$ al tempo iniziale. Analogamente i pointer successivi random_numbers_scenario[i], rappresentano le altre variabili stocastiche w_i necessarie per ottenere i successivi punti $S(t_i)$, fino al completamento del cammino (la cui lunghezza sarà pari a dim). La memorizzazione del vettore di oggetti Random_numbers, utilizzati nella costruzione del cammino si rivela particolarmente preziosa nel momento in cui si debba implementare un Monte Carlo con l'uso della variabile antitetica (vedi paragrafo 2.1.4.e).

Scheletro della classe (file .hpp)

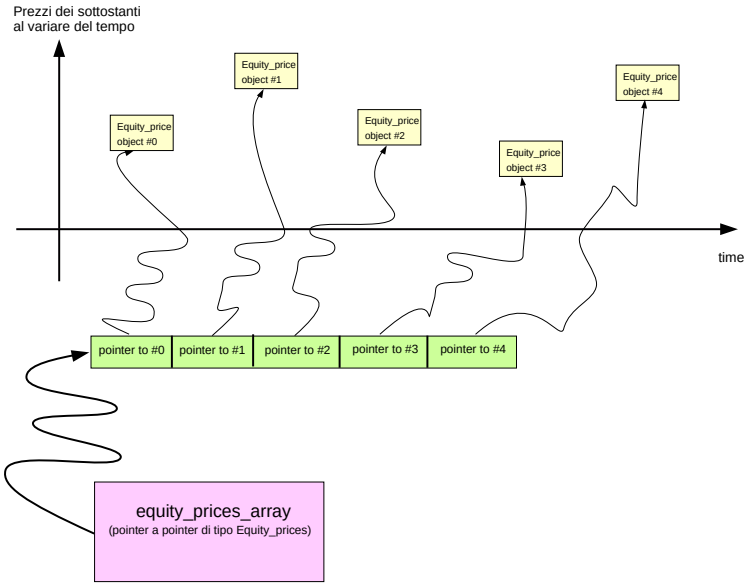

Figura 3.11: Schema grafico in cui viene rappresentato il significato di un pointer a pointer ad oggetti di tipo Equity_prices.

```
// ------------------ Begin path.hpp -----------------------------

class Path {

  private:
      Equity_prices *starting_point ;
      Equity_prices **equity_prices_scenario ;
      Random_numbers **random_numbers_scenario ;
      int dim ;

  public:
      // Constructors & destructors

      // Default constructor
      Path(void) ;

      Path(Equity_prices *starting_point_init,
           Schedule *schedule,
           Process_eq *process_eq) ;

      Path(Path *path) ;

      // Destructor
      virtual ~Path(void) ;
```

```
      Equity_prices *Get_starting_point(void) ;

      Equity_prices *Get_equity_prices(int i) ;

      Random_numbers *Get_random_numbers(int i) ;

      int Get_dim(vodi) ;

} ;

// ------------------ End path.hpp --------------------------------
```

Significato dei membri della classe

(1) Path(Equity_prices *starting_point_init, Schedule *schedule, Process_eq *process_eq) è il costruttore principale della classe Path.
Poiché i punti di un cammino (rappresentati nella nostra classe da equity_prices_scenario) si ottengano applicando iterativamente un processo stocastico, in modo da ottenere un nuovo punto a partire dal precedente, il costruttore in questione prende in input il punto di partenza (ovvero i prezzi delle azioni all'istante iniziale del cammino), il processo stocastico che ne governa l'evoluzione e l'insieme dei tempi che definiscono i punti temporali del cammino (ricordiamo infatti che il cammino di cui voliamo dare una rappresentazione è sempre formato da un insieme discreto di punti, di istantanee fotografiche prese in momenti differenti). Il costruttore utilizzerà il pointer starting_point_init per inizializzare starting_point e attraverso l'applicazione ricorsiva del processo stocastico process_eq, andrà a costruire il cammino formato dai punti specificati da schedule (ovviamente verranno generati effettivamente solo i punti di Schedule che appartengono al futuro rispetto alla data odierna presente nel campo time dell'oggetto puntato da starting_point_init).

(2) Path(Path *path): è il costruttore utilizzato per realizzare un cammino antitetico a partire dal quello contenuto in path. In sostanza l'algoritmo di generazione del cammino sarà identico a quello utilizzato nel costruttore di cui al punto precedente, con l'unica (importante) differenza che i numeri random necessari ad ottenere il nuovo punto equity_prices_scenario[i] partendo dal punto precedente equity_prices_scenario[i-1], non saranno generati ma verranno presi da random_numbers_scenario contenuto nell'oggetto puntato da path, invertendone i segni (vedi paragrafo 2.1.4.e).

(3) Get_starting_point(void): questa funzione restituisce il pointer al punto iniziale da cui si sviluppa il cammino.

(4) Get_equity_prices(int i): questa funzione restituisce il punto i-esimo del cammino.

(5) Get_random_numbers(int i): restituisce il pointer random_numbers_scenario[i], utilizzato per costuire l'oggetto puntato da equity_prices_scenario[i], partendo dall'oggetto puntato da equity_prices_scenario[i-1] [5].

Esercizio / laboratorio

Si implementino tutte le funzioni membro necessarie al corretto funzionamento della classe Path.

[5]Ovviamente per $i = 0$ si dovrà prendere come punto di partenza l'oggetto il cui indirizzo è contenuto in starting_point.

3.1.9 Anagrafica opzioni

- Descrizione: le classi sviluppate sotto, implementano il concetto di contratto opzionale. La struttura parte da una classe astratta (Contract_option) da cui vengono derivati due tronconi principali: Contract_equity_option (utilizzata come classe astratta per generare contratti opzionali scritti su una o più azioni) e Contract_IR_option (utilizzata come classe astratta per generare contratti scritti su tassi di interesse).

- Il diagramma di questo settore della libreria e' riportato nella figura sottostante.

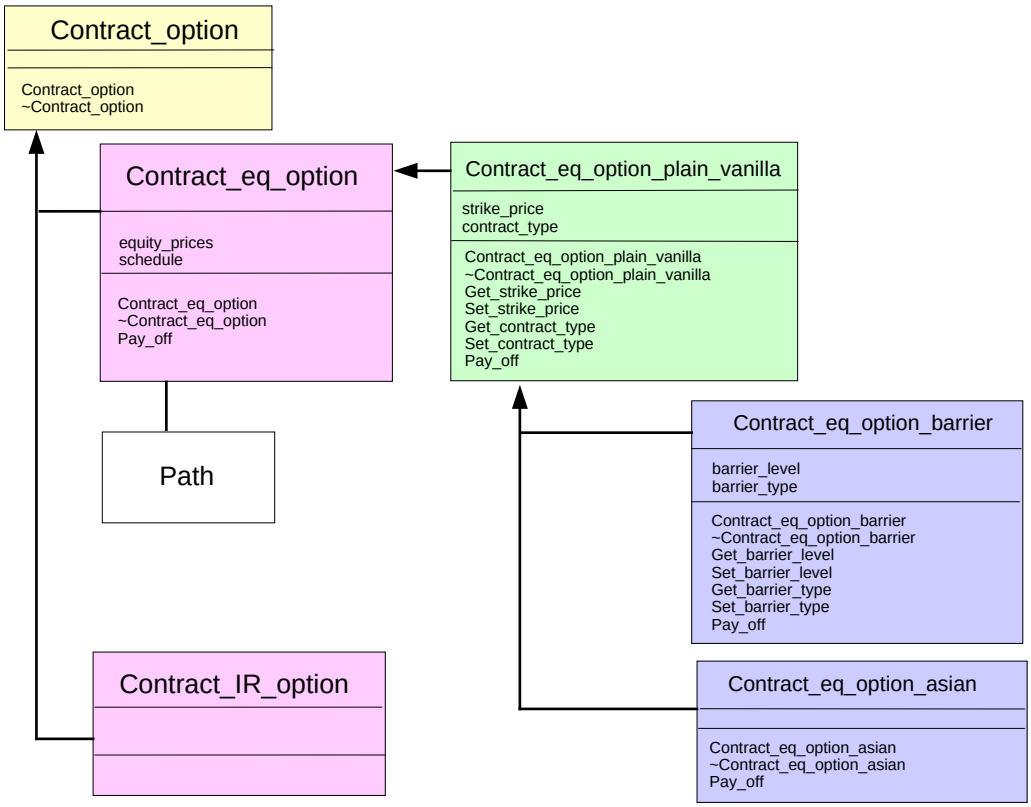

Figura 3.12: Schema anagrafica opzioni

3.1.9.a Classe: Contract_option

- Nome classe: Contract_option
- Significato: Classe astratta per gestire l'anagrafica delle opzioni.

Scheletro della classe (file .hpp)

```
// ------------------ Begin contract_option.hpp --------------------

class Contract_option {

  private:
      Schedule *schedule ;
      .
      .

  public:

      // Default constructor
      //
      Contract_option(void) ;

      Contract_option(Schedule *schedule_init) ;

      // Destructor
      //
      virtual ~Contract_option(void) {
      };


} ;


// ------------------ End contract_option.hpp ----------------------
```

Significato dei membri della classe

(1) schedule: è il pointer ad un oggetto di tipo Schedule contenente l'elenco delle date di fixing che caratterizzano il contratto opzionale.

3.1.9.b Classe: Contract_eq_option

- Nome classe: Contract_eq_option
- Significato: Classe astratta per gestire l'anagrafica delle opzioni su equity. Questa classe viene derivata da Contract_option.
  La classe Contract_eq_option dovrà avere tra i suoi membri un puntatore ad un oggetto di tipo Equity_prices che ne rappresenta il sottostante (in particolare l'anagrafica del o dei sottostanti sarà facilmente raggiungibile utilizzando il campo equity_description contenuto in ogni oggetto di tipo Equity_prices). Un altro elemento cruciale nella definizione di questa classe sarà la presenza di una funzione di pay-off che definisca il premio pagato a scadenza dall'opzione. Essendo Contract_eq_option una classe astratta il cui fine è semplicemente quello di fare da capostipite di una gerarchia di opzioni effettive, la funzione di pay-off all'interno di questa classe verrà dichiarata di tipo virtuale pura.

Scheletro della classe (file .hpp)

```
// ---------------- Begin contract_eq_option.hpp -------------------

class Contract_eq_option {

  private:
      Equity_prices *equity_prices ;

      .
  public:

      // Default constructor
      //
      Contract_eq_option(void) {
      };

      Contract_eq_option(Equity_prices *equity_prices_init,
                         Schedule *schedule_init) ;

      // Destructor
      //
      virtual ~Contract_eq_option(void) ;


      virtual double Pay_off(Path *path) = 0 ;
} ;
```

```
// ---------------- End contract_eq_option.hpp ---------------------
```

Significato dei membri della classe

(1) equity_prices: puntatore ad un oggetto di tipo Equity_prices. Questo membro permette di accedere al prezzo dei sottostanti ed alle relative anagrafiche (ricordiamo infatti che la classe Equity_prices contiene un pointer alla classe Equity_description).

(2) Pay_off(Path *path): funzione che restituisce il pay-off pagato dall'opzione a scadenza. In input dovrà ricevere il cammino del (o dei) sottostanti, path, in corrispondenza delle date di fixing specificate in schedule.

(3) Contract_equity_option(Equity_prices *equity_prices_init, Schedule *schedule_init): costruttore della classe Contract_equity_option, che inizializza le strutture: equity_prices e schedule.

Esercizio / laboratorio

Si implementino tutte le funzioni membro necessarie al corretto funzionamento della classe Contract_eq_option.

3.1.9.c Classe: Contract_eq_option_plain_vanilla

- Nome classe: Contract_eq_option_plain_vanilla
- Significato: Definisce la classe che descrive un'opzione *plain vanilla*. Questa classe viene derivata dalla classe padre Contract_eq_option

Scheletro della classe (file .hpp)

```
// ------------ Begin contract_eq_option_plain_vanilla.hpp ---------

class Contract_eq_option_plain_vanilla : public Contract_eq_option {

  private:
      double strike_price ;
      char contract_type ;

  public:

      // Default constructor
      //
      Contract_eq_option_plain_vanilla(void) {
      };

      // Constructor
      //
      Contract_eq_option_plain_vanilla(Equity *equity,
                             Schedule *schedule_init) ;

      // Destructor
      //
      virtual ~Contract_eq_option_plain_vanilla(void) ;

      // Pay off definition
      //
      virtual double Pay_off(Path *path) ;

} ;

// ------------ End contract_eq_option_plain_vanilla.hpp -----------
```

Significato dei membri della classe

(1) strike_price: variabile double contenente lo strike price dell'opzione.

(2) contract_type: variabile di tipo char per definire il tipo di contratto (ovvero 'P' per le opzioni di tipo put e 'C' per quelle di tipo call).

Esercizio / laboratorio

Si implementino tutte le funzioni membro necessarie al corretto funzionamento della classe Contract_eq_option_plain_vanilla. Infine partendo dalla classe per le opzioni di tipo *plain vanilla*, si derivino due classi per la gestione delle opzioni con barriera di tipo down & in, down & out, up & in e up & out e per le opzioni di tipo asiatico.

3.1.10 Metodi per il pricing di opzioni

- In questa sezione vengono esposte le classi necessarie a gestire i metodi (analitici e numerici) per la determinazione del prezzo di un'opzione. In particolare le classi riportate consentiranno di gestire le seguenti metodologie numeriche: Monte Carlo, Alberi binamiali e metodo alle differenze finite e le seguenti metodologie analitiche: formula chiusa (ad es. per le opzioni plain vanilla la classica formula di Black & Scholes) e formula chiusa approssimata (ad es. nel caso in cui fosse disponibile una formula analitica frutto di una qualche approssimazione).
  Il capostipite dei vari tipi di pricer, contiene dei riferimenti (pointer) al tipo di contratto da prezzare e al tipo di processo stocastico che domina l'evoluzione del sottostante. Sono infatti questi i due elementi comuni a tutti i vari tipi di pricer.

- Il diagramma di questo settore della libreria e' riportato nella figura sottostante.

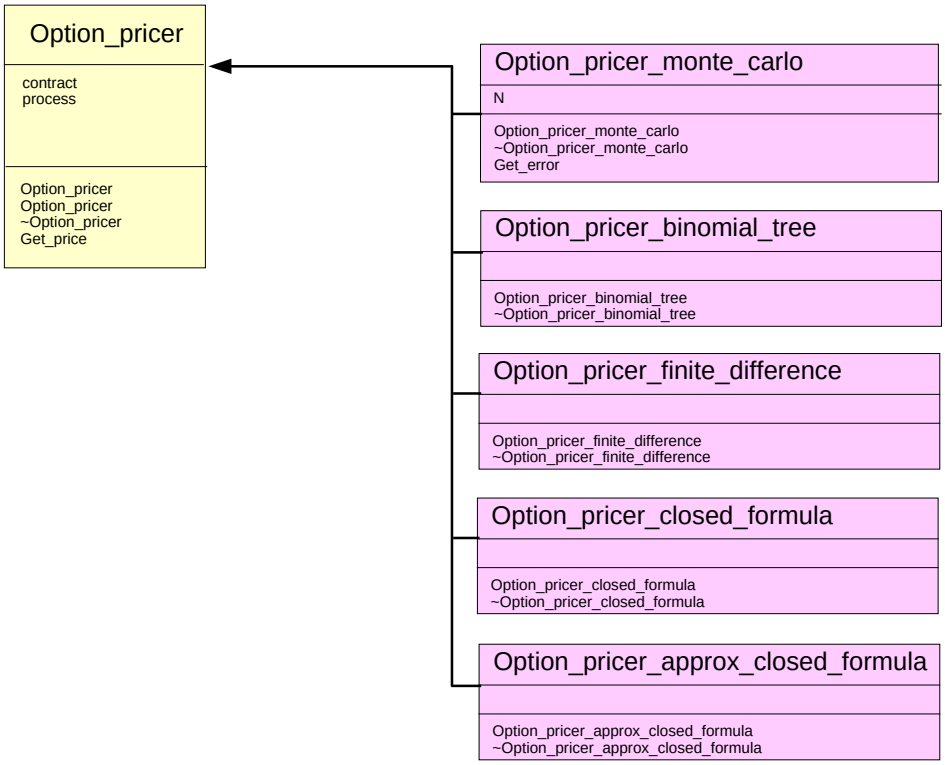

Figura 3.13: Schema pricer di opzioni

3.1.10.a Classe: Option_pricer

- Nome classe: Option_pricer
- Significato: Classe astratta da cui verranno derivati tutti i diversi tipi di pricer.

Scheletro della classe (file .hpp)

```
// ------------------ Begin option_pricer.hpp ----------------------

class Option_pricer {

  private:
      Contract_option *contract_option ;

      Process *process ;
      .
      .

  public:
      // Constructors & destructors

      // Default constructor
      //
      Option_pricer(void) {
      };

      // Constructor
      //
      Option_pricer(Contract_option *contract_option_init, Process *process_init) {
              contract_option = contract_option_init ;
              process = process_init ;
      };

      // Destructor
      //
      virtual ~Option_pricer(void) {
      };

      double Get_price(void) ;

} ;

// ------------------ End option_pricer.hpp ------------------------
```

Significato dei membri della classe

(1) contract: puntatore ad un oggetto di tipo Contract; rappresenta il contratto di cui si vule determinare il prezzo tramite il metodo in oggetto.

(2) process: puntatore ad un oggetto di tipo Process; rappresenta il processo stocastico che descrive l'evoluzione aleatoria del sottostante.

(3) Option_pricer(Contract *contract_init, Process *process_init): è il costruttore per la classe Pricer. Prende in input il contratto di cui si vuole determinare il prezzo ed il processo che governa l'evoluzione del sottostante il contratto. Nel costruttore si dovrà verificare la coerenza tra il contratto ed il processo passati in input, in particolare se il processo è di tipo multivariato, la dimensione del processo (ovvero il numero di sottostanti azionari che governa) deve essere uguale al numero di azioni presenti nell'oggetto puntato da equity_prices (più precisamente dato dal campo dim dell'oggetto di tipo Equity_prices puntato da equity_prices) che figura in contract (dopo aver provveduto all'opportuno cast esplicito a Contract_eq_option) [6].

(3) Get_price(void): funzione che restituisce il prezzo dell'opzione così come stimato dal pricer.

Esercizio / laboratorio

Relativamente al controllo a cui si è accennato nel punto (3) riportato sopra, si trovi un modo per stabilire (modificando opportunamente le classi Process) il sotto-tipo di processo puntato da process, prima di eseguire indiscriminatamente un eventuale cast esplicito. Questo al fine di poter gestire in maniera sicura l'operazione di cast che figura nella stringa (3.2). Un analogo discorso può essere ripetuto per la conversione di contract al sottotipo Contract_eq_option ed in generale per ogni situazione in cui si debba ricorrere ad un cast esplicito.

[6] In altri termini nel caso di un processo multivariato:

$$((\text{Contract\_eq\_option } *)\text{contract}) \rightarrow \text{Get\_equity\_prices}() \rightarrow \text{Get\_dim}() \,, \tag{3.1}$$

deve essere uguale a:

$$((\text{Process\_eq\_lognormal\_multivariate } *\text{process})) \rightarrow \text{Get\_dim}() \,. \tag{3.2}$$

3.1.10.b Classe: Option_pricer_monte_carlo

- Nome classe: Option_pricer_monte_carlo
- Significato: Classe per gestire il pricing con il metodo Monte Carlo. Tale classe costruirà N cammini (secondo l'algoritmo definito in 2.1.4.b) su ciascuno dei quali verrà calcolato il pay-off. La media e la standard deviation forniranno il valore e l'errore sulla stima del prezzo dell'opzione.

Scheletro della classe (file .hpp)

```
// -------------- Begin option_pricer_monte_carlo.hpp --------------

class Option_pricer_monte_carlo : public Option_pricer {

  private:
      // Number of simulation
      //
      int N ;
      .
      .

  public:

      // Default constructor
      //
      Option_pricer_monte_carlo(void) :  Option_pricer() {
      };

      Option_pricer(Contract_option *contract_option_init,
                    Process *process_init,
                    int N_init) ;

      // Destructor
      //
      virtual ~Option_pricer_monte_carlo(void) {
      };


      double Get_error(void) ;


} ;


// -------------- End option_pricer_monte_carlo.hpp ----------------
```

Significato dei membri della classe

(1) N: numero di simulazioni Monte Carlo.

(2) Option_pricer(Contract_option *contract_option_init, ...): il costruttore di questa classe è deputato a realizzare la simulazione Monte Carlo descritta nel paragrafo 2.1.4.
Di fatto l'algortimo verrà implementato reiternado N volte i seguenti passi:

- Si campiona lo spazio dei cammini stocastici $S(t)$ generando un oggetto di tipo Path. A tal fine si utilizzerà il costruttore di Path che consente di realizzare un nuovo cammino partendo da:
(1) un dato punto iniziale (in pratica il pointer equity_prices contenuto nell'oggetto contract_option, estraibile dopo aver eseguito un opportuno cast esplicito di contract_option al tipo Contract_eq_option);
(2) da un set di date definite dal pointer ad un oggetto Schedule che definisce le date di fixing del contratto opzionale (e precisamente il pointer Contract_option::schedule);
(3) dal pointer al processo stocastico che governa l'evoluzione del o dei sottostanti (passato al costruttore del pricer tramite il pointer process_init).
- Si valuta il pay-off lungo il cammino generato tramite la funzione di pay-off contenuta in contract_option (accedibile dopo aver eseguito un cast esplicito di questo pointer al tipo Contract_eq_option).
- Si cumula il pay-off calcolato (scontato ad oggi) ed il suo quadrato, in due variabili.

Una volta completato il ciclo sarà sufficiente calcolare media e varianza per avere una stima del prezzo dell'opzione con il relativo errore.

(3) Get_error(void): funzione che restituisce l'errore commesso nella stima Monte Carlo del prezzo (espresso percentualmente in maniera relativa rispetto al valore dell'opzione).

Esercizio / laboratorio

Si implementino tutte le funzioni membro necessarie al corretto funzionamento della classe Option_pricer_monte_carlo.

3.1.10.c Classe: Option_pricer_binamial_tree

- Nome classe: Option_pricer_binamial_tree
- Significato: Classe per gestire il pricing con il metodo dell'albero binomiale.

Scheletro della classe (file .hpp)

```
// -------------- Begin option_pricer_binomial_tree.hpp ------------

class Option_pricer_binomial_tree  : public Option_pricer {

  private:
      .
      .

  public:

      // Default constructor
      //
      Option_pricer_binomial_tree(void) :  Option_pricer() {
      };

      // Destructor
      //
      virtual ~Option_pricer_binomial_tree(void) ;
} ;


// -------------- End option_pricer_binomial_tree.hpp --------------
```

Esercizio / laboratorio

Si completi la classe Option_pricer_binamial_tree e si implementino tutte le funzioni membro necessarie al suo corretto funzionamento. In particolare si faccia riferimento al paragrafo 2.2.6 per quanto riguarda l'algoritmo da implementare.

3.1.10.d Classe: Option_pricer_finite_difference

- Nome classe: Option_pricer_finite_difference
- Significato: Classe per gestire il pricing con il metodo delle differenze finite.

Scheletro della classe (file .hpp)

```
// ------------- Begin option_pricer_finite_difference.hpp ---------

class Option_pricer_finite_difference  : public Option_pricer {

  private:
      .
      .

  public:

      // Default constructor
      //
      Option_pricer_finite_difference(void) :  Option_pricer() {
      };

      // Destructor
      virtual ~Option_pricer_finite_difference(void) {
      };


} ;


// ------------- End option_pricer_finite_difference.hpp -----------
```

Esercizio / laboratorio

Si completi la classe Option_pricer_finite_difference e si implementino tutte le funzioni membro necessarie al suo corretto funzionamento. In particolare si faccia riferimento al capitolo 2.3 per quanto riguarda l'algoritmo da implementare.

3.1.10.e Classe: Option_pricer_exact_closed_formula

- Nome classe: Option_pricer_exact_closed_formula
- Significato: Classe per gestire il pricing tramite formula chiusa.

Scheletro della classe (file .hpp)

```
// ------------- Begin option_pricer_closed_formula.hpp ------------

class Option_pricer_closed_formula  : public Option_pricer {

  private:
      .
      .

  public:
      // Constructors & destructors

      // Default constructor
      Option_pricer_closed_formula(void) :  Option_pricer() {
      };

      // Destructor
      virtual ~Option_pricer_closed_formula(void) {
      };


} ;

// ------------- End option_pricer_closed_formula.hpp --------------
```

Esercizio / laboratorio

Si completi la classe Option_pricer_exact_closed_formula e si implementino tutte le funzioni membro necessarie al suo corretto funzionamento. In particolare nel caso in cui il contratto sottostante sia di tipo *plain vanilla*, si implementi la formula chiusa di Black & Scholes illustrata nel paragrafo 1.3.5.

3.1.10.f Classe: Option_pricer_approx_closed_formula

- Nome classe: Option_pricer_approx_closed_formula
- Significato: Classe per gestire il pricing tramite formula approssimata

Scheletro della classe (file .hpp)

```
// ---------- Begin option_pricer_approx_closed_formula.hpp --------

class Option_pricer_approx_closed_formula  : public Option_pricer {

  private:
      .
      .

  public:
      // Constructors & destructors

      // Default constructor
      Option_pricer_approx_closed_formula(void): Option_pricer() {
      };

      // Destructor
      virtual ~Option_pricer_approx_closed_formula(void) {
      };


} ;


// ---------- End option_pricer_approx_closed_formula.hpp ----------
```

Esercizio / laboratorio

Si completi la classe Option_pricer_approx_closed_formula e si implementino tutte le funzioni membro necessarie al suo corretto funzionamento. In particolare nel caso in cui il contratto sottostante sia di tipo asiatico, si implementi la formula chiusa approssimata illustrata nel capitolo 1.5.

3.1.11 Schema complessivo della libreria (limitatamente al pricing di opzioni su equity)

Lo schema generale della libreria è definito in figura 3.14 . I collegamenti presenti nel grafico indicano le relazioni tra i differenti oggetti. Ad esempio, la linea direzionale che connette Equity_description a Yield_curve, indica che la classe Equity_description contiene un riferimento (e precisamente un puntatore) a Yield_curve. Alcune classi minori ed alcuni collegamenti sono stati omessi per semplificare il diagramma.
Lo schema complessivo non fa altro che riassumere in una visione di insieme le singole macroaree che abbiamo illustrato nei paragrafi precedenti. Partendo dalla classe Pricer (che può essere di 5 differenti sotto-tipi a seconda della metodologia di pricing scelta), questa contiene due riferimenti: uno al contratto opzionale tramite un pointer Contract_option (il cui sotto-tipo determina effettivamente la tipologia del contratto in oggetto) ed uno al processo che governa l'evoluzione del sottostante il contratto (il processo viene a sua volta scelto all'interno di una gerarchia, di cui il processo lognormale, rappresentato da Process_eq_lognormal, è un esempio). La classe Contract_option contiene poi un pointer ad un oggetto di tipo Schedule (che ne definisce le date di fixing) ed un pointer ad un oggetto di tipo Equity_prices. Quest'ultimo contiene il prezzo del o dei sottostanti all'istante attuale nonché i riferimenti alle relative anagrafiche. L'anagrafica di un'azione è poi in relazione con due altre classi: quella che descrive la curva dei tassi risk free e la classe deputata a gestire la superficie di volatilità che caratterizza l'azione.
Nel caso in cui il pricer sia di tipo Monte Carlo, la simulazione consisterà nella creazione di N cammini (su cui si valuterà media e varianza). I cammini sono generati attraverso la reiterazione del processo stocastico (necessariamente di tipo Process_eq [7]) lungo le date specificate dall'oggetto schedule contenuto in contract. Ogni cammino conterrà un riferimento (pointer) al punto di partenza (e precisamente ad un oggetto di tipo Equity_prices con il/i prezzo/i della/delle azione/i al tempo iniziale) ed un vettore di pointer (di dimensione dim) ad oggetti di tipo Equity_prices rappresentanti l'evoluzione stocastica del/dei sottostanti. Su ogni singolo cammino generato si effettuerà la valutazione del pay-off tramite la funzione Pay_off presente nella classe Contract_eq_option e nelle sue derivate (si noti a questo proposito il collegamento tra la classe Contract_eq_option [8] e Path).

[7] Sarà quindi necessario eseguire un cast esplicito del pointer process ad un pointer di tipo Process_eq.

[8] Anche in questo caso sarà quindi necessario eseguire un cast esplicito del pointer contract dal tipo Contract_option al tipo Contract_eq_option.

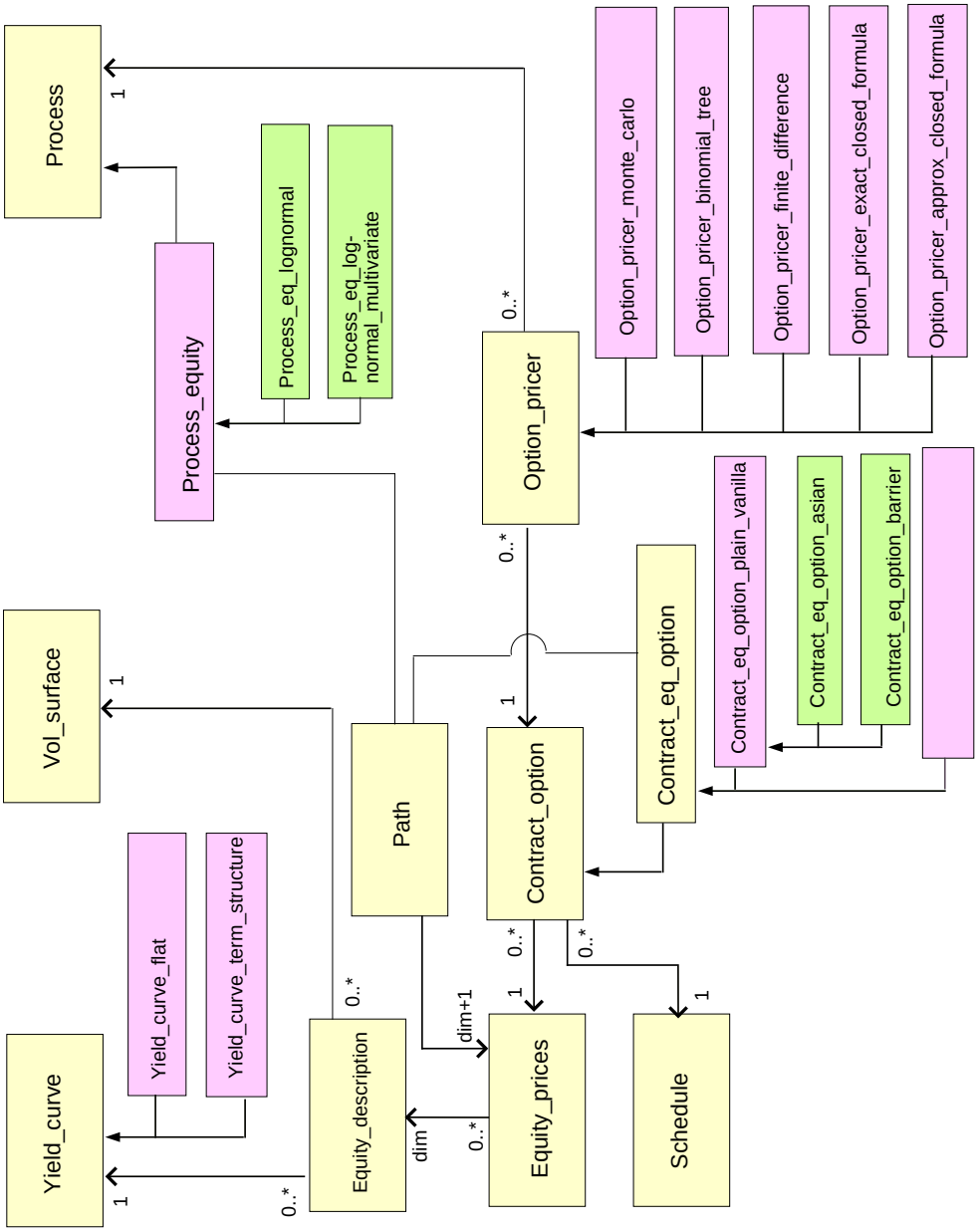

Figura 3.14: Schema generale delle libreria per il pricing di opzioni su equity.

3.1.12 Schema di flusso per il pricing di un'opzione scritta su di un sottostante azionario)

Lo schema di flusso da seguire per il pricing di un'opzione scritta su di un sottostante azionario è riportata in fig. 3.15. Tale schema andrà quindi implementato in un eventuale file main che andrà poi compilato utilizzando la libreria costruita nei paragrafi precedenti.

Schema di flusso per il pricing (main)

(1) Allocazione curva dei tassi

- Scopo: allocazione di un oggetto Yield_curve (di tipo flat o con una struttura a termine completa) al fine di definire una curva dei tassi risk free.
- Input esterno: il tasso risk free con eventuale struttura a termine.
- Link ad oggetti precedentemente allocati: nessuno.
- Propedeuticità: nessuna.

(2) Allocazione curva di volatilità

- Scopo: allocazione di un oggetto di tipo Vol_surface al fine di definire una curva di volatilità.
- Input esterno: la volatilità del sottostante del contratto opzionale.
- Link ad oggetti precedentemente allocati: nessuno
- Propedeuticità: nessuna.

(3) Allocazione anagrafica dell'azione

- Scopo: allocazione di un oggetto di tipo Equity_description al fine di definire l'anagrafica del sottostante su cui è scritta l'opzione.
- Input esterno: isin code, nome per esteso, currency di riferimento e dividend yield del titolo azionario.
- Link ad oggetti precedentemente allocati: la curva dei tassi di cui al punto (1) e la curva di volatilità di cui al punto (2).
- Propedeuticità: punti (1) e (2).

(4) Prezzo azionario attuale

- Scopo: allocazione di un oggetto di tipo Equity_prices al fine di definire il prezzo corrente del sottostante su cui è scritto il contratto opzionale.
- Input esterno: prezzo attuale dell'azione e istante temporale (espresso in termini di year fraction rispetto ad una data di riferimento standard) a cui tale prezzo si riferisce .

  - Link ad oggetti precedentemente allocati: l'anagrafica dell'azione di cui al punto (3).
  - Propedeuticità: punto (3).

(5) Date di fixing del contratto opzionale

  - Scopo: allocazione di un oggetto di tipo Schedule al fine di definire il set delle date di fixing che caratterizzano l'opzione (ad es. nel caso di una call plain vanilla, tale set sarà costituito unicamente dalla data di maturity dell'opzione).
  - Input esterno: l'insieme delle date di fixing (espresse in termini di year fraction rispetto ad una data di riferimento standard).
  - Link ad oggetti precedentemente allocati: nessuno.
  - Propedeuticità: nessuna.

(6) Anagrafica del contratto opzionale

  - Scopo: allocazione di un oggetto il cui tipo è derivato dalla classe Contract_eq_option (ad es. Contract_eq_option_plain_vanilla, Contract_eq_option_asian, Contract_eq_option_barrier ecc.) al fine di definire l'anagrafica dell'opzione di cui si desidera calcolare il prezzo.
  - Input esterno: le caratteristiche contrattuali dell'opzione (ad es. per una plain vanilla: lo strike price e il tipo di opzione call/put).
  - Link ad oggetti precedentemente allocati: il prezzo del sottostante di cui al punto (4) e le date di fixing del contratto di cui al punto (5).
  - Propedeuticità: punti (4) e (5).

(7) Processo stocastico

  - Scopo: allocazione di un oggetto di tipo Process_eq_lognormal al fine di definire il processo stocastico lognormale che governa l'evoluzione stocastica di un'azione.
  - Input esterno: nessuno.
  - Link ad oggetti precedentemente allocati: nessuno.
  - Propedeuticità: nessuna.

(8) Pricer

  - Scopo: allocazione di un oggetto di tipo Option_pricer al fine calcolare il prezzo dell'opzione. A seconda del tipo di pricer allocato si potrà determinare il prezzo dell'opzione tramite formula chiusa

(se esiste), albero binomiale, metodo alle differenze finite o simulazione Monte Carlo.
L'esecuzione del pricing verrà effettuata richiamando nel main la funzione Option_pricer::Get_price() che restituirà il prezzo. Nel caso del Monte Carlo sarà possibile avere una stima dell'errore commesso tramite la funzione Option_pricer_monte_carlo::Get_error().

- Input esterno: parametri specifici per il tipo di pricer allocato (ad es. nel caso del Monte Carlo: numero totale di simulazioni e flag per definire l'uso o meno della variabile antitetica).
- Link ad oggetti precedentemente allocati: il processo stocastico di cui al punto (7) e l'anagrafica dell'opzione di cui al punto (6).
- Propedeuticità: punti (6) e (7).

In figura 3.15 è riportato un diagramma che mostra graficamente lo svilupparsi dei vari punti con le loro propedeuticità.

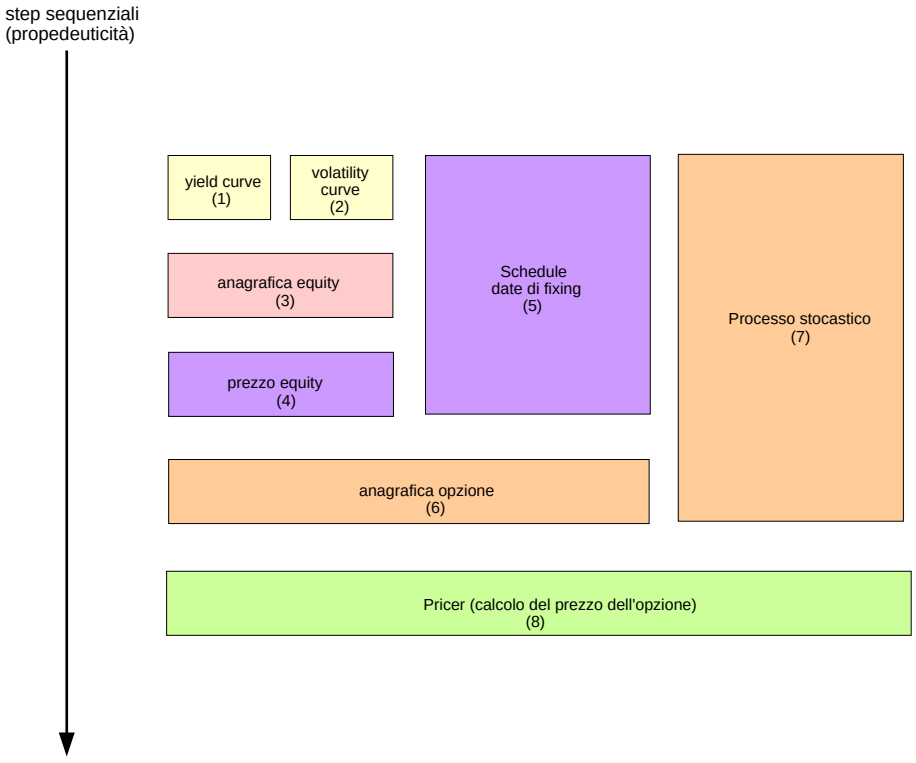

Figura 3.15: Schema di un ipotetico programma main, con l'allocazione dei vari oggetti e le relative propedeuticità.

3.2 Link in Excel di una libreria C++

Un punto essenziale nel lavoro di un ingegnere finanziario è collegare le librerie scritte in C++ all'interno di un foglio excel (il quale farà da front end).
A titolo di esempio, vedremo come costruire un collegamento tra un foglio excel ed una libreria dll in C, al fine di calcolare il prezzo di un'opzione asiatica (potremo pensare a questo proposito di utilizzare la formula analitica approssimata ottenuta nel capitolo 1.5).
L'attività può essere suddivisa nei seguenti passi:

- Definire un foglio Excel. Nel worksheet principale (che chiamiamo main), le righe della prima colonna riportano le seguenti informazioni:
  - B:1 = prezzo attuale del sottostante
  - B:2 = tasso di interesse privo di rischio
  - B:3 = volatilità
  - B:4 = strike price dell'opzione
  - B:5 = tipo opzione (0 = call, 1 = put)
  - B:6 = numero date di rilevazione
  - B:7 = prima data di rilevazione
  - B:8 = seconda data di rilevazione
  - B:n = ultima data di rilevazione, dove n e' pari a 7+B:6.

- Dentro il foglio Excel, accedere all'applicazione VBA, aggiungedo un modulo dal nome "module_pricing.bas" con il codice riportato nelle pagine successive. Tale codice verrà richimato tramite un pulsante aggiunto al foglio excel.

- Salvare il foglio excel con il relativo codice

- Definire un progetto per la creazione di una libreria di tipo dll. Nel progetto verrà inserito un file dll_interface.cpp (vedi codice riportate sotto).

- All'interno del file dll_interface.cpp, viene definita una funzione che corrisponde a quella dichiarata e richiamata all'interno del codice Visual Basic di Excel (precisamente nel nostro esempio la funzione Interface_Pricing). Questa funzione avrà come input i vettori e le variabili provenienti da Visual Basic (vedi codice riportato sotto). L'unica accortezza a questo punto è convertire le strutture dati provenienti da Visual Basic in strutture dati accessibili da C. Tale compito viene assolto ad es. dalla funzione Get_vector(). Per quanto riguarda le

variabili, non sono necessari accorgimenti particolari, si noti solo che le strutture provenienti da Visual Basic sono sempre costituite da puntatori. Inoltre è bene ricordare che l'analogo VBA delle variabili int di C è il Long (su questo punto ritorneremo alla fine del capitolo).

- Compilare la libreria dll

3.2.1 file dll_interface.cpp

```
// -------------------- Begin dll_interface.cpp --------------------

#include <windows.h>
#include <stdio.h>
#include <ole2.h>


#pragma pack(1)

BOOL APIENTRY Dll_test( HANDLE hModule, DWORD ul_reason_for_call, LPVOID lpReserved )
{
return TRUE;
}



// EXAMPLES

extern "C"  int __stdcall Interface_Pricing_asian (

       // dati di mercato
       //
       double  * vba_S_0,
       double  * vba_r,
       double  * vba_vol,

       // anagrafica contratto
       //
       double  * vba_strike_price,
       int * vba_contract_type,
       int * vba_num_fixing_date,
       LPSAFEARRAY FAR  * vba_fixing_date,

       // Output
       //
       double * vba_result
       ) {


          // Dichiarazioni variabili
          //
          double S_0;
          double r ;
          double vol;

          double strike_price ;
          int contract_type ;
          int num_fixing_date ;
```

```
          double *fixing_date ;

          // Output
          //
          double result ;


          S_0 = *vba_S_0 ;
          r = *vba_r ;
          vol = *vba_vol ;
          strike_price = *vba_strike_price ;
          contract_type = *vba_contract_type ;
          num_fixing_date = *vba_num_fixing_date ;

          fixing_date= Convert_vector(vba_fixing_date) ;

          result = Pricing_asian(S_0, r, vol, strike_price, contract_type,
                              num_fixing_date, fixing_date) ;

          *vba_result = result ;

          delete fixing_date ;

}

double * Convert_vector(LPSAFEARRAY FAR *pointer) {

          double *pdata ;
          pdata = (double *)((*pointer)->pvData);
          double *vector = new double[((*pointer)->rgsabound->cElements)] ;

          for (int i=0;i<((*pointer)->rgsabound->cElements);i++) {
               vector[i] = *pdata ;
          pdata++ ;
          }

          return vector ;

}

// -------------------- End dll_interface.cpp ----------------------
```

3.2.2 file module_pricing.bas

```
// -------------------- Begin module_pricing.bas -------------------

' CODICE VISUAL BASIC


Attribute VB_Name = "Module_pricing_asian"


Declare Sub Interface_Pricing_asian Lib "lib_excel_c.dll" ( _
                     vba_S_0 As Double, _
                     vba_r As Double, _
                     vba_vol As Double, _
                     vba_strike_price As Double, _
                     vba_contract_type As Long, _
                     vba_num_fixing_date As Long, _
                     vba_fixing_date() As Double, _
                     vba_result As Double)

Sub Pricing_asian()

  Dim vba_S_0 As Double
  Dim vba_r As Double
  Dim vba_vol As Double
  Dim vba_strike_price As Double
  Dim vba_contract_type As Long
  Dim vba_num_fixing_date As Long
  Dim vba_fixing_date() As Double
  Dim vba_result As Double
  Dim j as Integer
  Dim wk_main As String


  wk_main = "main"

  ' Reading data
  '
  vba_S_0 = Worksheets(wk_main).Cells(1, 2).value
  vba_r = Worksheets(wk_main).Cells(2, 2).value
  vba_vol = Worksheets(wk_main).Cells(3, 2).value
  vba_strike_price = Worksheets(wk_main).Cells(4, 2).value
  vba_contract_type = Worksheets(wk_main).Cells(5, 2).value
  vba_num_fixing_date = Worksheets(wk_main).Cells(6, 2).value

  ReDim vba_fixing_date(vba_num_fixing_date - 1) As Double

  For j = 0 To vba_num_fixing_date - 1
     vba_num_fixing_date(j) = Worksheets(wk_main).Cells(7 + j, 2).value
  Next j
```

```
  ' Stop

  ' Call Interface function
  '
  Interface_Pricing_asian Lib vba_S_0, _
      vba_r, _
      vba_vol, _
      vba_strike_price, _
      vba_contract_type, _
      vba_num_fixing_date, _
      vba_fixing_date(), _
      vba_result

  Worksheets(wk_main).Cells(1, 4) = vba_result

End Sub

// -------------------- End module_pricing.bas ----------------------
```

Si noti che la chiamata alla funzione di C, fatta dal modulo di excel, utilizza come variabili intere delle variabili di tipo Long, questo perché gli Integer di Visual Basic non corrispondono, come lunghezza, alle variabili int di C. Più precisamente si dovranno usare a tale scopo le variabili Long di Visual Basic.

Capitolo 4

Laboratorio ed esercitazioni pratiche

Laboratorio - introduzione

Nei successivi capitolo verranno presentate alcune esercitazioni, con l'obiettivo di testare e collaudare la libreria di pricing illustrata nel capitolo 3.1. In tali esercitazioni l'obiettivo consisterà nell'effettuare il pricing di opzioni esotiche, scrivendo inizialmente il codice necessario e successivamente sviluppando l'esercitazione lungo le linee guida indicate.

Gli esercizi sono divisi in due gruppi distini: quelli della serie "a" sono caratterizzati da un focus specifico sugli aspetti implementativi e teorici; la serie "b" è invece più "general purpose" e non richiede necessariamente l'uso di un linguaggio di programmazione ad oggetti per il suo svolgimento.

4.1 Esercizio serie a.1

4.1.1 Introduzione

Il presente capitolo è dedicato all'esposizione e discussione di una sessione di laboratorio / esercitazione relativa al metodo Monte Carlo presentato nel capitolo 2.1.
Nel primo paragrafo verrà presentato il contratto di una opzione esotica di cui ci proponiamo di calcolare il prezzo tramite una simulazione Monte Carlo. Nel par. 4.1.3 vengono presentate una serie di considerazioni. Infine nel par. 4.1.4 vengono esposti i punti su cui focalizzare l'esercitazione.
L'obiettivo della presente esercitazione non è solo quello di svolgere un semplice esercizio di programmazione, implementando un Monte Carlo, ma di mostrare piuttosto come l'integrazione di un approccio numerico con considerazioni qualitative e analitiche è in grado di portare a migliori risultati.

4.1.2 Definizione del problema

Si consideri un contratto opzionale con le seguenti caratteristiche:

- Opzione con esercizio europeo (ovvero il diritto può essere fatto valere unicamente alla data di scadenza);
- Sia T la data di maturità e $t = 0$ la data attuale. Il sottostante dell'opzione è un'azione, il cui prezzo al tempo t sia indicato con $S(t)$;
- Supponiamo che il prezzo S dell'azione segua un processo log-normale, con μ costante e volatilità σ costante (l'usuale modello utilizzato in finanza);
- Il pay-off pagato a scadenza sia:

$$Max\left[0, S(T) - (E + n\,K)\right] \ ; \tag{4.1}$$

  dove: E e K sono due costanti positive e n rappresenta la percentuale di volte in cui, nell'intervallo $(0, T)$, il ritorno giornaliero del sottostante, $S(t+1)/S(t) - 1$, sia stato inferiore alla costante $-L$.
  In altri termini il pay-off è quello di una normale call option in cui però lo strike price non è costante ma è tanto più grande quanto maggiore è la percentuale di volte in cui il sottostante ha avuto una performance giornaliera negativa superiore in valore assoluto a L.

4.1.3 Discussione qualitativa

Dal punto di vista della nomenclatura, si tratta di un'opzione call con *strike* variabile ed esercizio di tipo europeo. È ovviamente un'opzione *path dependent*, in quanto il pay-off dipende non solo dal valore del sottostante alla

maturity date, ma anche dalla sua storia precedente.
È interessante osservare che nel limite in cui $L \to \infty$ (o $K \to 0$) il pay-off si riduce a quello di una semplice opzione *call plain vanilla*. In tal caso il prezzo dell'opzione in ogni istante è dato dalla formula di BS ed in base al lemma di Ito, il prezzo stesso dell'opzione segue un processo stocastico di Ito. Ciò significa che per il mercato l'unica cosa che conti nell'evoluzione di questo prezzo è il valore che esso assume oggi (processo markoviano).
Più interessante è quanto accade per L (o K) finito. In tale situazione l'opzione diventa path dependent, in altri termini la storia del sottostante è importante per prezzare correttamente il contratto. È ovvio che a parte situazioni limite, non è possibile ottenere una formula chiusa per il pricing di questo contratto. Bisogna quindi ricorrere ad una simulazione Monte Carlo, che è l'obiettivo della presente esercitazione. È comunque possibile definire delle disuguaglianze a cui il prezzo dell'opzione deve sempre ubbidire. Infatti è abbastanza ovvio che il toccare la barriera negativa $-L$ nei ritorni, ha l'effetti di alzare ogni volta lo strike price di una quantità pari a K. Se indichiamo con:

- $\mathcal{C}$ un generico cammino da $(0, T)$ (soddisfacente il vincolo $S(t = 0) = S_0$);
- $n_{\mathcal{C}}$ la percentuale di volte nell'intervallo $(0, T)$ in cui l'azione, lungo il cammino $\mathcal{C}$, ha avuto un ritorno giornaliero minore di $-L$;
- Pay-off$_{\text{vanilla}}(\mathcal{C}, \bar{E})$ il pay-off su $\mathcal{C}$ di una opzione call europea con strike $\bar{E}$;
- Pay-off$(\mathcal{C})$ il pay-off dell'opzione in esame;

allora:

$$\text{Pay-off}(\mathcal{C}) = \text{Pay-off}_{\text{vanilla}}(\mathcal{C}, E + n_{\mathcal{C}}\, K)\ ; \tag{4.2}$$

e quindi:

$$\text{Pay-off}_{\text{vanilla}}(\mathcal{C}, E + K) \leq \text{Pay-off}(\mathcal{C}) \leq \text{Pay-off}_{\text{vanilla}}(\mathcal{C}, E)\ ; \tag{4.3}$$

ne consegue che il prezzo P dell'opzione soddisfa la seguente disuguaglianza:

$$P_{\text{call vanilla}}(E + K) \leq P \leq P_{\text{call vanilla}}(E)\ ; \tag{4.4}$$

Ritorniamo ora alla questione del processo stocastico seguito dal prezzo dell'opzione. Dimostreremo la seguente affermazione:

Proposizione Il processo stocastico seguito dal prezzo dell'opzione definita sopra, non è markoviano.
Dimostriamo prima il seguente lemma:
Lemma: Supponiamo di conoscere l'evoluzione del prezzo dell'opzione, $P(t)$,

per ogni $t \in (0,T)$. Indichiamo con $S(t)$ il prezzo del sottostante e con $n(t)$ il rapporto tra il numero di volte in cui i ritorni giornalieri del sottostante siano stati inferiori a $-L$ nell'intervallo $(0,t)$ ed il numero di giorni in $(0,T)$. Allora a partire dalla conoscenza di $P(t)$ è possibile determinare le funzioni $n(t)$ e $S(t)$.

Dim. Dalle caratteristiche del contratto, è immediato dedurre che il prezzo dell'opzione, in ogni istante t, è funzione unicamente delle variabili: t, $S(t)$ e $n(t)$, ovvero $P(t) = P(t,S,n)$.

Dimostriamo il lemma procedendo per induzione. Per $t = 0$, $n(0) = 0$ in base alla definizione data. Inoltre dall'equazione $P(0,S(0),0) = P(0)$ possiamo ricavare, invertendola, $S(0)$.

Supponiamo ora di conoscere n e S al tempo $t-1$ e proponiamoci di ricavare $n(t)$ e $S(t)$.

Al tempo t, essendo noto il prezzo dell'opzione, $P(t)$, possiamo scrivere l'equazione:

$$P\left(t, S(t), n(t)\right) = P(t) \tag{4.5}$$

dove:

$$n(t) = \begin{cases} n(t-1) + 1/D & \text{se } [S(t)/S(t-1) - 1] < -L \\ n(t-1) & \text{diversamente} \end{cases}$$

e D rappresenta il numero di giorni contenuti nell'intervallo $(0,T)$.

Allora poiché $n(t-1)$ e $S(t-1)$ sono noti, $n(t)$ è una funzione della sola variabile $S(t)$. Pertanto l'eq. (4.5) si riduce ad un'equazione nella sola incognita $S(t)$. È altresì facile dimostrare che tale funzione è monotona decrescente. L'equazione (4.5) può allora essere invertita per trovare $S(t)$ (e perciò anche $n(t)$). Rimane quindi dimostrata la tesi.

Passiamo ora alla dimostrazione della proposizione principale.

Dim. Supponiamo di considerare un tempo $t \in (0,T)$ e sia $P(t)$ il prezzo dell'opzione. Se ora consideriamo l'istante temporale $t + dt$, il prezzo dell'opzione varierà unicamente in base alle variazioni di prezzo del sottostante. Quindi se desideriamo poter determinare la distribuzione probabilistica di P all'istante $t+dt$, dobbiamo necessariamente inferire il prezzo del sottostante S al tempo t per poi poter costruire la distribuzione probabilistica di S al tempo $t + dt$ (a tal riguardo ricordiamo che $\log(S)$ segue un processo di Wiener generalizzato). Una volta nota tale distribuzione è quasi immediato ricavare dalla funzione $P(t,S,n)$ la distribuzione statistica del prezzo dell'opzione all'istante $t + dt$. Ovviamente per poter dedurre il valore di $S(t)$ dal prezzo $P(t)$ dobbiamo necessariamente conoscere $n(t)$ e questo è possibile solo conoscendo la serie storica dei prezzi di P (vedi lemma precedente). Ecco quindi che la distribuzione statistica di $P(t+dt)$ viene a dipendere non solo da $P(t)$ ma anche da tutti i prezzi precedenti.

Rimane quindi dimostrato che:

- il prezzo dell'opzione in ogni istante dipende dalla storia del sottostante;
- se riguardiamo il prezzo dell'opzione, come una variabile stocastica, questa non solo non segue un processo di Ito, ma addirittura non è markoviana.
- In particolare l'evoluzione stocastica futura del prezzo dell'opzione al tempo t dipende dalla serie di tutti i prezzi nell'intervallo $(0, t)$. Esistono quindi sul mercato oggetti finanziari, il cui prezzo non segue un processo markoviano.
- Ugualmente, per tale opzione vale il principio di non arbitraggio, nel senso che non è possibile realizzare un guadagno superiore al tasso di mercato *risk free*, assumendo una posizione qualsiasi su questo strumento. Ciò dimostra che l'ipotesi di non markovianità non viola, necessariamente, il principio di non arbitraggio.

4.1.4 Obiettivi dell'esercitazione

Si richiede di:

(A) Sviluppare un programma in C che implementi una simulazione Monte Carlo per il pricing dell'opzione esotica descritta nel paragrafo 4.1.2. Tale programma deve prendere come input un file con la seguente struttura:

```
# Numero di simulazioni Monte Carlo
N =

# Valore dell'azione all'istante iniziale
S_0 =

# Valore del tasso risk free
r =

# Valore della volatilita', su base annua, del sottostante
sigma =

# Barriera per la performance giornaliera
L =

# Strike price dell'opzione nel caso in cui la
# barriera non venga mai toccata
E =
```

```
# Incremento dello strike price nel caso in cui la
# barriera venga toccata il 100% delle volte
K =

# Intervallo di tempo espresso in numero di giorni
T =
```

le linee che iniziano con il carattere # devono essere considerate di commento. Come suggerimento sui valori numerici da utilizzare per i primi *run*, si considerino i seguenti:

$$\begin{aligned} &N = 100.000\ , \\ &S_0 = 100\,, \\ &r = 4\%\,, \\ &\sigma = 30\%\,, \\ &L = 4\%\,, \\ &E = 100\,, \\ &K = 10\,, \\ &T = 100\ \text{ business days}\,. \end{aligned} \tag{4.6}$$

(B) Implementare il metodo Monte Carlo descritto nel paragrafo 2.1.4, in un primo momento attraverso la discretizzazione del processo stocastico definita dall'equazione (2.14) e successivamente utilizzando la versione migliorata (2.18).
i) Si verifichi quindi che per $K = 0$ o $L \to \infty$ il programma riproduca correttamente il *pricing* di una opzione call europea con *strike* E;
ii) partendo dal punto precedente, si discutano nei due casi (discretizzazione (2.14) e versione migliorata (2.18)), le differenze nelle stime del prezzo dell'opzione (eventualmente anche cambiando i parametri T e N).
Per tutti i successivi punti si faccia sempre riferimento alla formula (2.18).

(C) Al fine di ridurre gli errori statistici, si utilizzino: la tecnica della variabile antitetica (vedi par. 2.1.4.e) e la tecnica della variabile di controllo (vedi par. 2.1.4.e). Relativamente alla scelta della variabile di controllo, è possibile effettuare, nel presente caso, una scelta migliore che non sia quella di una *call plain vanilla* con *strike price* pari ad E? Si richiede di riportare nell'output del programma la stima del prezzo dell'opzione: i) senza i miglioramenti ottenuti con l'utilizzo delle due tecniche; ii) la stima con l'utilizzo della sola variabile antitetica; iii)

la stima con l'utilizzo della sola variabile di controllo; vi) la stima ottenuta con entrambe le tecniche.
Calcolare e discutere la riduzione degli errori statistici (ovvero le deviazioni standard sui pay-off) ottenuta con l'uso delle due tecniche.

(D) Verificare che il pricing ottenuto soddisfi la disuguaglianza (4.4).

(E) Calcolare il numero medio di volte in cui il ritorno giornaliero del sottostante tocca il limite $-L$; determinare anche la deviazione standard di questa stima. Discutere come tali stime si comportino al crescere di T.

(F) Stimare il valore medio dello *strike price* $E + n\,K$ e determinarne la deviazione standard. Discutere come tali grandezze si comportino al crescere di T.

(G) Discutere qualitativamente quale sia l'effetto sul *pricing*, considerando valori di T (maturità dell'opzione) molto alti. In particolare si osservino i dati ottenuti ai punti (E) e (F), con valori grandi di T, e li si utilizzi per inferire una valutazione analitica approssimata (valida nel limite $T \to \infty$) del prezzo dell'opzione. Verificare poi numericamente la validità di tale approssimazione.

I risulti ottenuti devono essere presentati sotto forma di breve discussione scritta, con l'ausilio eventuale di grafici e tabelle.

4.2 Esercizio serie a.2

4.2.1 Introduzione

Il presente capitolo è dedicato all'esposizione e discussione di una sessione di laboratorio / esercitazione relativa al metodo Monte Carlo presentato nel capitolo 2.1.
Nel primo paragrafo verrà presentato il contratto di una opzione esotica di cui ci proponiamo di calcolare il prezzo tramite una simulazione Monte Carlo. Nel par. 4.2.3 vengono presentate una serie di considerazioni. Infine nel par. 4.2.4 vengono esposti i punti su cui focalizzare l'esercitazione.
L'obiettivo della presente esercitazione non è solo quello di svolgere un semplice esercizio di programmazione, implementando una simulazione Monte Carlo, ma di mostrare piuttosto come l'integrazione di un approccio numerico con considerazioni qualitative e analitiche è in grado di portare a risultati migliori.

4.2.2 Definizione del problema

Si consideri un contratto opzionale con le seguenti caratteristiche:

- Opzione con esercizio europeo (ovvero il diritto può essere fatto valere unicamente alla data di scadenza);
- Sia T la data di maturità e $t = 0$ la data attuale. Il sottostante dell'opzione sia un'azione, il cui prezzo al tempo t venga indicato con $S(t)$;
- Supponiamo che il prezzo $S(t)$ dell'azione segua un processo log-normale, con μ costante e volatilità σ costante (ovvero l'usuale modello browniano utilizzato in finanza);
- Il pay-off pagato a scadenza sia:

$$\text{Pay-off}_{\text{Rev. cliquet}} = \text{Max}\left[L, H + \frac{1}{m}\sum_{i=1}^{m}\text{Min}\left(\frac{S_i - S_{i-1}}{S_{i-1}}, 0\right)\right] \ , \quad (4.7)$$

  dove m e' il numero di intervalli (periodi) considerati; S_i è il prezzo del sottostante alla fine del periodo i-esimo (t_{i-1}, t_i); S_0 e lo spot price (ovvero il prezzo ad oggi dell'azione); L è il "floor" globale (usualmente posto a zero) e corrisponde al minimo ammontare che l'investitore riceverà alla data di scadenza; H è il massimo coupon pagabile dall'opzione.
  Si tratta quindi di un'opzione il cui pay-off risulta limitato all'interno della banda: $[L, H]$.
  L'opzione definita dal pay-off riportato sopra, prende il nome di "reverse cliquet".

4.2.3 Alcune considerazioni

Si osservi che all'interno dell'equazione (4.7), il termine $\frac{S_i - S_{i-1}}{S_{i-1}}$ rappresenta la performance dell'azione relativamente al periodo i-esimo. Perciò la sommatoria presente nell'equazione che definisce il pay-off, va interpretata come la media delle performance negative dell'azione. Queste ultime, vanno calcolate su un intervallo di lunghezza T/m.
Il termine "reverse" che figura nel nome dell'opzione, sta ad indicare che il termine costante H viene decrementato di una quantità legata all'andamento dell'azione (ovvero la media delle performance negative).
Le opzioni reverse cliquet, sono un tipico esempio di opzioni path dependent. È inutile sottolineare che non esistono formule chiuse esatte per la valutazione di questo tipo di opzione.

4.2.4 Obiettivi dell'esercitazione

Si richiede di:

(A) Sviluppare un programma ad oggetti in C++ che implementi una simulazione Monte Carlo per il pricing dell'opzione esotica descritta nel paragrafo 4.2.2. Tale programma deve prendere come input i seguenti dati:

```
Numero di simulazioni Monte Carlo
N = 10.000 - 100.000

Valore dell'azione all'istante iniziale
S_0 = 100

Valore del tasso risk free
r = 9 %

Valore della volatilita', su base annua, del sottostante
sigma = 30 %

Minimo garantito dall'opzione
L = 0 %

Massimo coupon pagabile dall'opzione
H = 4 %

Intervallo di tempo su cui calcolare le performance azionarie:
T/m = 1/12 anno

Numero di intervalli
```

```
m = 1, 3, 6, 9, 12, 15, 18, 60, 120
```

(B) Implementare il metodo Monte Carlo descritto nel paragrafo 2.1.4 utilizzando la versione migliorata (2.18).

(C) Si ottenga una formula chiusa per il prezzo dell'opzione nel caso in cui $m = 1$. Verificare poi numericamente la validità di tale derivazione.

(D) Mantenendo fissa la lunghezza dell'intervallo su cui si calcolano le performance dell'azione, ovvero T/m, discutere il caso in cui il numero di intervalli m diventi molto alto (al limite infinito). In particolare, si ricavi una formula asintotica per il prezzo dell'opzione, valida nel limite $m \to \infty$. Verificare poi numericamente, con l'ausilio delle simulazioni Monte Carlo, la validità di tale approssimazione.

(E) In base ai risultati di cui al punto (C), si esaminino delle possibili scelte per la variabile di controllo, al fine di migliorare la stima delle simulazioni Monte Carlo.

(F) Allo scopo di ridurre gli errori statistici, si utilizzino: la tecnica della variabile antitetica [1] e la tecnica della variabile di controllo [2] sulla base delle indicazioni emerse nel punto precedente.
Si richiede di valutare il prezzo dell'opzione: i) senza i miglioramenti ottenuti con l'utilizzo delle due tecniche; ii) la stima con l'utilizzo della sola variabile antitetica; iii) la stima con l'utilizzo della sola variabile di controllo; vi) la stima ottenuta con entrambe le tecniche.
Calcolare e discutere la riduzione degli errori statistici (ovvero le deviazioni standard sui pay-off) ottenuta con l'uso delle due tecniche.

I risulti ottenuti devono essere presentati sotto forma di breve discussione scritta, con l'ausilio eventuale di grafici [3] e tabelle.

[1] Vedi par. 2.1.4.e

[2] Vedi par. 2.1.4.e

[3] Ad esempio mostrando l'andamento del prezzo dell'opzione al variare di m, e la sua convergenza ai due limiti estremi $m = 1$ e $m = \infty$, ricavati ai punti (C) e (D).

4.3 Esercizio serie a.3

4.3.1 Introduzione

Il presente capitolo è dedicato all'esposizione e discussione di una sessione di laboratorio / esercitazione relativa al metodo Monte Carlo presentato nel capitolo 2.1.
Nel primo paragrafo verrà presentato il contratto di una opzione esotica di cui ci proponiamo di calcolare il prezzo tramite una simulazione Monte Carlo. Nel par. 4.3.3 vengono presentate una serie di considerazioni. Infine nel par. 4.3.4 vengono esposti i punti su cui focalizzare l'esercitazione.
L'obiettivo della presente esercitazione non è solo quello di svolgere un semplice esercizio di programmazione, implementando una simulazione Monte Carlo, ma di mostrare piuttosto come l'integrazione di un approccio numerico con considerazioni qualitative e analitiche è in grado di portare a risultati migliori.

4.3.2 Definizione del problema

Si consideri un contratto opzionale con le seguenti caratteristiche:

- Opzione con esercizio europeo (ovvero il diritto può essere fatto valere unicamente alla data di scadenza);

- Sia T la data di maturità e $t = 0$ la data attuale. Il sottostante dell'opzione sia un'azione, il cui prezzo al tempo t venga indicato con $S(t)$;

- Supponiamo che il prezzo $S(t)$ dell'azione segua un processo log-normale, con μ costante e volatilità σ costante (ovvero l'usuale modello browniano utilizzato in finanza);

- Il pay-off pagato a scadenza sia:

$$\text{Pay-off}_{\text{Average perf.}} = \text{Max}\left[0, X - E\right] \ , \tag{4.8}$$

  dove X è la somma delle performance del sottostante sugli m intervalli equispaziati di ampiezza T/m, in cui è suddiviso l'intevallo temporale $(0, T)$, da oggi alla data di maturità dell'opzione. Ovvero:

$$X = \sum_{i=1}^{m} \frac{S(t_i) - S(t_{i-1})}{S(t_{i-1})} \ , \tag{4.9}$$

  dove $t_i = i\ T/m$ per $i = 0, ..., m$.

4.3.3 Alcune considerazioni

Il pay-off dell'opzione definita sopra è identico al classico pay-off di una call con l'importante differenza che la variabile X risulta essere data dalla somma delle performance del sottostante sugli m intervalli in cui è suddiviso l'intervallo $(0, T)$. Si noti anche che lo strike price è definito in percentuale essendo X un numero puro.
L'opzione è chiaramente path dependent, in quanto X è definito in base al percorso seguito dal sottostante durante la vita dell'opzione (in particolare dipende dai valori assunti dal sottostante in corrispondenza delle date di fixing $\{t_i\}$.

4.3.4 Obiettivi dell'esercitazione

Si richiede di:

(A) Sviluppare un programma ad oggetti in C++ che implementi una simulazione Monte Carlo per il pricing dell'opzione esotica descritta nel paragrafo 4.3.2. Tale programma deve prendere come input i seguenti dati:

```
Numero di simulazioni Monte Carlo
            N = 10.000
            N = 20.000
            N = 40.000

Valore dell'azione all'istante iniziale
            S_0 = 100

Valore dello strike price (in %)
            E = 4 %

Valore del tasso risk free
            r = 4 %

Valore della volatilita', su base annua, del sottostante
            sigma = 30 %

Durata dell'opzione T pari a:
            T = 0.5 anni
            T = 1.0 anni
            T = 1.5 anni
            T = 2.0 anni
            T = 2.5 anni
```

```
                T = 3.0 anni
                T = 3.5 anni
                T = 4.0 anni
                T = 10.0 anni

Numero di intervalli m pari a:
                m = 2
                m = 5
                m = 10
                m = 20
                m = 50
                m = 100
                m = 200
```

In particolare si implementi il metodo Monte Carlo descritto nel paragrafo 2.1.4 utilizzando la versione migliorata (2.18).

(B) Produrre un insieme di grafici che mostrino l'andamento del prezzo dell'opzione per differenti valori di T (in particolare si faccia riferimento ai valori di T specificati nel punto A) al variare di m (ovvero in ascissa il grafico riporterà il tempo a maturità T, in ordinata il valore dell'opzione e ciascun grafico si riferirà ad un differente valore di m; i valori di m per cui produrre i grafici sono specificati nel punto A).

(C) Produrre un insieme analogo di grafici (come nel punto B) che mostrino l'andamento dell'errore nella stima del prezzo dell'opzione per differenti valori di T e m (fissato $N = 10.000$).

(D) Discutere il comportamento dei grafici e la loro interpretazione finanziaria quando il numero di intervalli m diventa molto grande (in linea di principio infinito). Fornire un'interpretazione teorica dei risultati ottenuti per m molto grande, in particolare:
i) è possibile derivare una formula analitica asintotica per il calcolo del prezzo dell'opzione nel limite $m \to \infty$? ii) si confrontino i risultati ottenuti dalla formula asintotica con le stime Monte Carlo.

(E) Allo scopo di ridurre gli errori statistici, si utilizzi: la tecnica della variabile antitetica [4]. In particolare si richiede di:
i) valutare il prezzo dell'opzione ed il relativo errore utilizzando la tecnica della variabile antitetica;
ii) calcolare e discutere la riduzione degli errori statistici (ovvero le deviazioni standard sui pay-off) ottenuta tramite l'uso della variabile antitetica.

[4] Vedi par. 2.1.4.e

(F) i) Ricavare una formula analitica esatta per l'opzione nel caso in cui $m = 1$. ii) Confrontare i risultati ottenuti dalla formula esatta con le stime Monte Carlo per $m = 1$.

(G) Rispetto al metodo standard della variabile di controllo [5], in cui:

$$x_{\text{MC new}} = x_{\text{MC}} + y - y_{\text{MC}} \ , \tag{4.10}$$

si sviluppi una variante della formula sopra riportata al fine di sfruttare pienamente l'informazione contenuta nella variabile di controllo.

(H) Sfruttando il risultato ottenuto nel punto (F), allo scopo di ridurre gli errori statistici, si ricorra alla tecnica della variabile di controllo, nella sua versione migliorata, derivata nel punto precedente, utilizzando come variabile il prezzo della seguente opzione:

$$\text{Pay-off}_{\text{Average perf. m=1}} = \text{Max}\left[0, \frac{S(T) - S(t_0)}{S(t_0)} - E\right] \ , \tag{4.11}$$

In particolare si richiede di:
i) di valutare il prezzo dell'opzione ed il relativo errore utilizzando la variabile di controllo introdotta sopra.
ii) calcolare e discutere la riduzione degli errori statistici (ovvero le deviazioni standard sui pay-off) ottenuta tramite l'uso della variabile di controllo al variare di m.

I risulti ottenuti devono essere presentati sotto forma di breve discussione scritta, con l'ausilio eventuale di grafici e tabelle.

[5] Vedi par. 2.1.4.e

4.4 Esercizio serie a.4

4.4.1 Introduzione

Il presente capitolo è dedicato all'esposizione e discussione di una sessione di laboratorio / esercitazione relativa al metodo Monte Carlo presentato nel capitolo 2.1.
Nel primo paragrafo verrà presentato il contratto di una opzione esotica di cui ci proponiamo di calcolare il prezzo tramite una simulazione Monte Carlo. Nel par. 4.4.3 vengono presentate una serie di considerazioni. Infine nel par. 4.4.4 vengono esposti i punti su cui focalizzare l'esercitazione.
L'obiettivo della presente esercitazione non è solo quello di svolgere un semplice esercizio di programmazione, implementando una simulazione Monte Carlo, ma di mostrare piuttosto come l'integrazione di un approccio numerico con considerazioni qualitative e analitiche è in grado di portare ad una migliore comprensione del problema e da un punto di vista più operativo a stime migliori.

4.4.2 Definizione dell'opzione "Counter Positive Performance"

Si consideri un contratto opzionale con le seguenti caratteristiche:

- Opzione con esercizio europeo (ovvero il diritto può essere fatto valere unicamente alla data di scadenza);

- Sia T la data di maturità e $t = 0$ la data attuale. Il sottostante dell'opzione sia un'azione, il cui prezzo al tempo t venga indicato con $S(t)$;

- Supponiamo che il prezzo $S(t)$ dell'azione segua un processo log-normale, con μ costante e volatilità σ costante (ovvero il modello browniano base utilizzato in finanza);

- Il pay-off pagato a scadenza sia:

$$\text{Pay-off}_{\text{counter perf.}} = K \cdot \text{Max}\left[\frac{n}{m} - p_0, 0\right] \ , \tag{4.12}$$

  dove K è una costante espressa in unità di valuta (ad es. euro), $m+1$ è il numero delle date di fixing, p_0 è un numero compreso nell'intervallo $[0, 1]$ e può essere interpretato come una percentuale e n è il numero di volte in cui lungo il cammino si verifica che $S(t_i)$ sia maggiore o uguale a $S(t_{i-1})$ con $i = 1, ...m$. Si suppone che gli intervalli (t_{i-1}, t_i) siano tutti equispaziati e di ampiezza T/m, dove T è la durata complessiva dell'opzione, ovvero: $t_i = i\ T/m$ per $i = 0, ..., m$.

4.4.3 Alcune considerazioni

L'opzione descritta sopra, che potremmo chiamare "Counter positive performance", è chiaramente di tipo path dependent in quanto il pay-off corrisposto alla data di scadenza è definito in base al percorso seguito dal sottostante durante la vita dell'opzione (in particolare dipende dai valori assunti dal sottostante in corrispondenza delle date di fixing $\{t_i\}$. Il rapporto n/m che figura nella formula di pay-off rappresenta la percentuale di volte in cui l'azione ha avuto una performance positiva lungo il cammino. Le performance sono calcolate relativamente ai vari intervalli (t_{i-1}, t_i). La costante p_0 può essere interpretata come una sorta di strike price in riferimento alla percentuale di cui sopra; ovvero: se la percentuale di performance positive, $p = n/m$, lungo il cammino è maggiore di p_0 viene pagato un premio pari alla differenza tra p e p_0, diversamente non viene pagato nulla. È naturale perciò considerare valori di p_0 compresi nell'intervallo $[0, 1]$, interpretando questa grandezza come una percentuale.

4.4.4 Obiettivi dell'esercitazione

Si richiede di:

(A) Sviluppare una libreria finanziaria in base alla traccia fornita nel capitolo 3.1, limitatamente alla parte di option pricing su singoli sottostanti azionari e per il solo metodo Monte Carlo. In particolare si implementi il metodo Monte Carlo descritto nel paragrafo 2.1.4, utilizzando la versione migliorata (2.18).

(B) Introdurre nella libreria una classe per descrivere l'anagrafica dell'opzione descritta nel paragrafo 4.4.2. Sviluppare quindi un main (ovvero un programma) in grado di calcolare una stima del prezzo dell'opzione, con il relativo errore [6].

(C) Testare il programma ottenuto utilizzando come dati di input i seguenti valori:

```
Numero di simulazioni Monte Carlo
             N = 10.000
             N = 20.000
             N = 40.000

Valore dell'azione all'istante iniziale
             S_0 = 100
```

[6] Nota bene: le stime dell'errore nel prezzo dell'opzione vanno espresse in forma percentuale sul prezzo stesso, ovvero: $\epsilon = \frac{\text{errore assoluto}}{\text{stima prezzo}}\%$.

```
Valore della costante K (in euro)
            K = 1 euro

Valore della costante p0 (in euro)
            p0 = 0.5

Valore del tasso risk free
            r = 4 %

Valore della volatilita', su base annua, del sottostante
            sigma = 30 %

Durata dell'opzione T pari a:
            T = 1.0 anni

Numero di intervalli m pari a:
            m = 1
            m = 5
            m = 10
            m = 20
            m = 50
            m = 100
            m = 200
```

(D) Allo scopo di ridurre gli errori statistici, si utilizzi: la tecnica della variabile antitetica [7]. In particolare si richiede di:
i) valutare il prezzo dell'opzione ed il relativo errore utilizzando la tecnica della variabile antitetica;
ii) calcolare e discutere la riduzione degli errori statistici (ovvero le deviazioni standard sui pay-off) ottenuta tramite l'uso della variabile antitetica.

(E) Produrre un grafico che mostri l'andamento del prezzo dell'opzione per differenti valori di m (in particolare si faccia riferimento ai valori di m specificati nel punto (C). Il grafico riporterà in ascissa il numero di date di fixing, $m + 1$, e in ordinata il valore dell'opzione.

(F) Produrre un insieme analogo di grafici (come nel punto (E)) che mostrino l'andamento dell'errore nella stima del prezzo dell'opzione per differenti valori di m (fissato ad es. $N = 10.000$).

[7] Vedi par. 2.1.4.e

(G) Discutere il comportamento dei grafici e la loro interpretazione finanziaria quando il numero di intervalli m diventa molto grande (in linea di principio infinito). Fornire un'interpretazione teorica dei risultati ottenuti per m molto grande, in particolare:
i) è possibile derivare una formula analitica asintotica per il calcolo del prezzo dell'opzione nel limite $m \to \infty$? ii) si confrontino i risultati ottenuti dalla formula asintotica con le stime Monte Carlo.

(H) i) Ricavare una formula analitica esatta per l'opzione nel caso in cui $m = 1$. ii) Confrontare i risultati ottenuti dalla formula esatta con le stime Monte Carlo per $m = 1$.

(I) Ottenere una formula semi-analitica/semi-numerica per il calcolo del prezzo dell'opzione per un valore generico di m. Si confrontino quindi i risultati ottenuti con le stime Monte Carlo.

(L) Rispetto al metodo standard della variabile di controllo [8], in cui:

$$x_{\text{MC new}} = x_{\text{MC}} + y - y_{\text{MC}} \ , \tag{4.13}$$

si sviluppi una variante della formula sopra riportata al fine di sfruttare pienamente l'informazione contenuta nella variabile di controllo.

(M) Sfruttando il risultato ottenuto nel punto (H), allo scopo di ridurre gli errori statistici, si ricorra alla tecnica della variabile di controllo, nella sua versione migliorata, derivata nel punto precedente, utilizzando come variabile il prezzo dell'opzione "Counter Positive Performance" con $m = 1$.
In particolare si richiede di:
i) di valutare il prezzo dell'opzione ed il relativo errore utilizzando la variabile di controllo introdotta sopra.
ii) calcolare e discutere la riduzione degli errori statistici (ovvero le deviazioni standard sui pay-off) ottenuta tramite l'uso della variabile di controllo al variare di m.

I risulti ottenuti devono essere presentati sotto forma di breve discussione scritta, con l'ausilio eventuale di grafici e tabelle. Queste ultime devono avere come finalità quella di dare un supporto all'interpretazione ed alla discussione dei risultati, in generale quindi è buona norma evitare di riportare numerose tabelle e grafici se questi non servono all'economia generale della relazione. Inoltre tutti i grafici e le tabelle vanno commentati ed integrati nella relazione.

[8] Vedi par. 2.1.4.e

4.5 Esercizio serie a.5

4.5.1 Introduzione

Il presente capitolo è dedicato all'esposizione e discussione di una sessione di laboratorio / esercitazione relativa al metodo Monte Carlo presentato nel capitolo 2.1.
Nel primo paragrafo verrà presentato il contratto di una opzione esotica di cui ci proponiamo di calcolare il prezzo tramite una simulazione Monte Carlo. Nel par. 4.5.3 vengono presentate una serie di considerazioni. Infine nel par. 4.5.4 vengono esposti i punti su cui focalizzare l'esercitazione.
L'obiettivo della presente esercitazione non è solo quello di svolgere un semplice esercizio di programmazione, implementando una simulazione Monte Carlo, ma di mostrare piuttosto come l'integrazione di un approccio numerico con considerazioni qualitative e analitiche è in grado di portare a risultati migliori.

4.5.2 Definizione del problema

Si consideri un contratto opzionale con le seguenti caratteristiche:

- Opzione con esercizio europeo (ovvero il diritto può essere fatto valere unicamente alla data di scadenza);
- Sia T la data di maturità e $t = 0$ la data attuale. Il sottostante dell'opzione sia un'azione, il cui prezzo al tempo t venga indicato con $S(t)$;
- Supponiamo che il prezzo $S(t)$ dell'azione segua un processo log-normale, con μ costante e volatilità σ costante (ovvero l'usuale modello browniano utilizzato in finanza);
- Il pay-off pagato a scadenza sia:

$$\text{Pay-off}_{\text{Difference with corridor}} = \text{Max}\left[X, 0\right] \ , \tag{4.14}$$

  dove X è la somma delle variabili D_i definite sotto e legate all'andamento del sottostante sugli m intervalli equispaziati di ampiezza $\delta_t = T/m$, in cui è suddiviso l'intevallo temporale $(0, T)$, da oggi alla data di maturità dell'opzione. Ovvero:

$$X = \sum_{i=1}^{m} D_i \ , \tag{4.15}$$

$$D_i = \text{Min}\left[\text{Max}\left[S(t_i) - S(t_{i-1}), l_i\right], u_i\right] \ , \tag{4.16}$$

dove:

$$l_i = S(t_{i-1}) \frac{\left(e^{\hat{r}\,\delta_t} - 1\right)}{k} , \tag{4.17}$$

$$u_i = S(t_{i-1}) \left(e^{\hat{r}\,\delta_t} - 1\right) k , \tag{4.18}$$

e $t_i = i\,T/m$ per $i = 0, ..., m$.
Le grandezze l_i e u_i rappresentano rispettivamente il valore minimo e massimo assunto dalla variabile D_i. D_i è legato alla differenza nel prezzo del sottostante calcolato sull'intervallo i-esimo, limitato inferiormente e superiormente da l_i e u_i.
$\hat{r}$ è un parametro specificato nel contratto derivato. Nel seguito, per semplicità, si supporrà sempre che $\hat{r}$ sia uguale al tasso risk free e quindi implicitamente che la curva dei tassi risk free sia piatta, ovvero che il tasso risk free non dipenda dalla scadenza considerata.
k è un parametro in grado di restringere o allargare il gap (l_i, u_i) e soddisfa sempre il seguente vincolo:

$$k >= 1 . \tag{4.19}$$

Si noti che per $k = 1$ il gap si stringe completamente e D_i è vincolato ad assumere il valore: $\left(e^{\hat{r}\,\delta_t} - 1\right)$; viceversa per $k = \infty$, D_i coincide con la variazione nel prezzo del sottostante sull'intervallo i-esimo.

4.5.3 Alcune considerazioni

Il pay-off dell'opzione definita sopra è identico al classico pay-off di una call con l'importante differenza che la variabile X risulta più complessa di quanto non avvenga in un'ordinaria plain vanilla option. Inoltre per $m > 1$, il contratto diventa di tipo path-dependent, in quanto X è definito in base al percorso seguito dal sottostante durante la vita dell'opzione (in particolare dipende dai valori assunti dal sottostante in corrispondenza delle date di fixing $\{t_i\}$).
Si noti come nel caso limite $k = +\infty$ e $m = 1$, il pay-off divenga:

$$\text{Pay-off}_{\text{difference with corridor}} = \text{Max}\left[S(t_1) - S(t_0), 0\right] , \tag{4.20}$$

ovvero sia quello di una call ordinaria con strike price uguale al valore di partenza dell'azione all'istante iniziale.

4.5.4 Obiettivi dell'esercitazione

Alla fine di ogni punto viene riportato quanto segue:

- **Propedeuticità.**

- **Grado di difficoltà: (Basso, Medio, Alto, Molto Alto)**
- **Tempo di sviluppo atteso: (Breve, Medio, Lungo)**
- **Richiesta ai fini dell'esame: (Obbligatorio, Facoltativo)**

Si richiede di sviluppare e risolvere i seguenti punti:

(A) *Pricing Monte Carlo*

(A-1) Sviluppare un programma ad oggetti in C++ che implementi una simulazione Monte Carlo per il pricing dell'opzione esotica descritta nel paragrafo 4.5.2. Tale programma deve prendere come input i seguenti dati:

```
Numero di simulazioni Monte Carlo
   N = 10.000
   N = 20.000
   N = 40.000

(in generale i valori riportati sopra sono solo indicativi. E' lasciata
allo studente la scelta di quali valori assumere per tale grandezza).

Valore dell'azione all'istante iniziale
   S_0 = 100

Valore dello strike price (in %)
   k = 1.0
   k = 1.1
   k = 1.2
   k = 1.3
   k = 1.4
   k = 1.5
   k = 1.6
   k = 1.7
   k = 1.8
   k = 1.9
   k = 2.0
   k = 10

(in generale i valori riportati sopra sono solo indicativi. E' lasciata
allo studente la scelta di quali valori assumere per tale grandezza).

Valore del tasso risk free
   r = 2 %
```

```
Valore della volatilita', su base annua, del sottostante
   sigma = 30 %

Durata dell'opzione T pari a:
   T = 0.5 anni
   T = 1.0 anni
   T = 1.5 anni
   T = 2.0 anni
   T = 2.5 anni
   T = 3.0 anni
   T = 3.5 anni
   T = 4.0 anni
   T = 10.0 anni

(in generale i valori riportati sopra sono solo indicativi. E' lasciata
allo studente la scelta di quali valori assumere per tale grandezza).


Numero di intervalli m pari a:
   m = 2
   m = 5
   m = 10
   m = 20
   m = 50
   m = 100
   m = 200

(in generale i valori riportati sopra sono solo indicativi. E' lasciata
allo studente la scelta di quali valori assumere per tale grandezza).
```

In particolare si implementi il metodo Monte Carlo descritto nel paragrafo 2.1.4 utilizzando: a) il metodo approssimato di Eulero e b) la formula integrale esatta (2.18).

Caratteristiche del punto:

- **Propedeuticità: nessuna.**
- **Grado di difficoltà: Medio**
- **Tempo di sviluppo atteso: Lungo**
- **Ai fini dell'esame: obbligatorio**

(A-2) Produrre un insieme di grafici che mostrino l'andamento del prezzo dell'opzione per differenti valori di T e k (in particolare si faccia

riferimento ai valori di T e k specificati nel punto A-1) al variare di m (ovvero in ascissa il grafico riporterà il tempo a maturità T o k, in ordinata il valore dell'opzione e ciascun grafico si riferirà ad un differente valore di m; i valori di m per cui produrre i grafici sono specificati nel punto A).

Caratteristiche del punto:

- **Propedeuticità: (A-1).**
- **Grado di difficoltà: Medio**
- **Tempo di sviluppo atteso: Lungo**
- **Ai fini dell'esame: obbligatorio**

(A-3) Produrre un insieme analogo di grafici (come nel punto A-2) che mostrino l'andamento dell'errore nella stima del prezzo dell'opzione per differenti valori di T/k e m (fissato $N = 10.000$).
N.B. L'errore va calcolato percentualmente ed in forma relativa, ovvero: errore MC assoluto / prezzo opzione, espresso in %.

Caratteristiche del punto:

- **Propedeuticità: (A-1).**
- **Grado di difficoltà: Medio**
- **Tempo di sviluppo atteso: Lungo**
- **Ai fini dell'esame: obbligatorio**

(B) *Formule di valutazione esatte*

(B-1) i) Ricavare una formula analitica esatta per l'opzione nel caso in cui $m = 1$ (con k e σ qualunque). ii) Confrontare i risultati ottenuti dalla formula esatta con le stime Monte Carlo per $m = 1$.

Caratteristiche del punto:

- **Propedeuticità: (A) per quanto riguarda il confronto con i dati Monte Carlo. Nessuna per quanto riguarda la derivazione della formula analitica.**
- **Grado di difficoltà: Medio/Alto**
- **Tempo di sviluppo atteso: Breve**
- **Ai fini dell'esame: Facoltativo**

(B-2) i) Ricavare una formula analitica esatta per l'opzione nel caso in cui $k = 1$ (con m e σ qualunque). ii) Confrontare i risultati ottenuti dalla formula esatta con le stime Monte Carlo per $k = 1$.

Caratteristiche del punto:

- **Propedeuticità: (A) per quanto riguarda il confronto con i dati Monte Carlo. Nessuna per quanto riguarda la derivazione della formula analitica.**

- **Grado di difficoltà: Medio**
- **Tempo di sviluppo atteso: Breve**
- **Ai fini dell'esame: Facoltativo**

(B-3) i) Ricavare una formula analitica esatta per l'opzione nel caso in cui $k = \infty$ (con m e σ qualunque). ii) Confrontare i risultati ottenuti dalla formula esatta con le stime Monte Carlo per $k = \infty$.

Caratteristiche del punto:

- **Propedeuticità: (A) per quanto riguarda il confronto con i dati Monte Carlo. Nessuna per quanto riguarda la derivazione della formula analitica.**
- **Grado di difficoltà: Basso**
- **Tempo di sviluppo atteso: Breve**
- **Ai fini dell'esame: Facoltativo**

(B-4) i) Ricavare una formula analitica esatta per l'opzione nel caso in cui $\sigma = 0$ (con m e k qualunque).

Caratteristiche del punto:

- **Propedeuticità: nessuna. Può essere d'aiuto aver risolto il punto (B-2).**
- **Grado di difficoltà: Medio/Basso**
- **Tempo di sviluppo atteso: Breve**
- **Ai fini dell'esame: Facoltativo**

(C) *Formule di valutazione approssimate*

(C-1) Discutere il comportamento dei grafici e la loro interpretazione finanziaria quando il numero di intervalli m diventa molto grande (in linea di principio infinito). Fornire un'interpretazione teorica dei risultati ottenuti per m molto grande, in particolare:
i) derivare una formula analitica asintotica per il calcolo del prezzo dell'opzione nel limite $m \to \infty$; ii) si confrontino i risultati ottenuti dalla formula asintotica con le stime Monte Carlo.

Caratteristiche del punto:

- **Propedeuticità: (A) per quanto riguarda il confronto con i dati Monte Carlo. Per la derivazione della formula analitica asintotica, si suggerisce di risolvere prima i punti (B-2) e (B-3).**
- **Grado di difficoltà: Molto alto**
- **Tempo di sviluppo atteso: Medio**
- **Ai fini dell'esame: Facoltativo**

(D) *Tecniche di riduzione dell'errore*

(D-1) Allo scopo di ridurre gli errori statistici, si utilizzi: la tecnica della variabile antitetica [9]. In particolare si richiede di:
i) valutare il prezzo dell'opzione ed il relativo errore utilizzando la tecnica della variabile antitetica;
ii) calcolare e discutere la riduzione degli errori statistici (ovvero le deviazioni standard sui pay-off) ottenuta tramite l'uso della variabile antitetica.

Caratteristiche del punto:

- **Propedeuticità: (A).**
- **Grado di difficoltà: Basso**
- **Tempo di sviluppo atteso: Medio**
- **Ai fini dell'esame: Obbligatorio**

(D-2) Rispetto al metodo standard della variabile di controllo [10], in cui:

$$x_{\text{MC new}} = x_{\text{MC}} + y - y_{\text{MC}} \ , \tag{4.21}$$

si sviluppi una variante della formula sopra riportata al fine di sfruttare pienamente l'informazione contenuta nella variabile di controllo.

Caratteristiche del punto:

- **Propedeuticità: Nessuna.**
- **Grado di difficoltà: Medio/Alto**
- **Tempo di sviluppo atteso: Medio**
- **Ai fini dell'esame: Facoltativo**

(D-3) Sfruttando il risultato ottenuto nei punti (B1) e/o (B2) e/o (B3), allo scopo di ridurre gli errori statistici, si ricorra alla tecnica della variabile di controllo, nella sua versione migliorata, derivata nel punto precedente. In particolare si richiede di:
i) di valutare il prezzo dell'opzione ed il relativo errore utilizzando la variabile di controllo.
ii) calcolare e discutere la riduzione degli errori statistici (ovvero le deviazioni standard sui pay-off) ottenuta tramite l'uso della variabile di controllo al variare di m.

Caratteristiche del punto:

- **Propedeuticità: (B-1) e/o (B-2) e/o (B-3) e (D-2).**
- **Grado di difficoltà: Basso**
- **Tempo di sviluppo atteso: Medio**

[9] Vedi par. 2.1.4.e
[10] Vedi par. 2.1.4.e

- **Ai fini dell'esame: Facoltativo**

(E) *Processi stocastici*

(E-1) Sviluppare una classe per gestire un processo alternativo a quello lognormale standard, in particolare tale processo è definito sostituendo nella formula (1.36), al posto della variabile gaussiana w a media nulla e varianza unitaria, una variabile stocastica alternativa z (sempre a media nulla e varianza unitaria), definita come segue:

$$z = \begin{cases} +1 & \text{con probabilità } 0.5 \\ -1 & \text{con probabilità } 0.5 \end{cases} \tag{4.22}$$

Il prezzo dell'azione è definito da:

$$S(\delta t) = S(0)\, e^{\left(r - \frac{\sigma^2}{2}\right)\delta t + \sigma\sqrt{\delta t}\, z}\,, \tag{4.23}$$

Il processo in questione ha natura binaria, nel senso che consente unicamente un movimento al ribasso o al rialzo dell'azione su un certo intervallo di tempo δt.

Si verifichi che la libreria sia strutturata in maniera tale da consentire l'utilizzo del nuovo processo senza apportare modifiche alla libreria (a parte naturalmente l'aggiunta e la definizione della nuova classe per la descrizione del processo binario). Si discuta quali siano i vantaggi di un tale risultato e se questo sia o meno ottenibile nell'ambito di una programmazione procedurale.

Caratteristiche del punto:

- **Propedeuticità: punto (A).**
- **Grado di difficoltà: Basso**
- **Tempo di sviluppo atteso: Basso**
- **Ai fini dell'esame: Facoltativo**

(E-2) Si consideri il pricing di un'opzione plain vanilla, spezzando l'intervallo $(0, T)$ (T essendo la data di maturità dell'opzione) in m intervalli. Effettuare la valutazione del prezzo dell'opzione plain vanilla con metodo Monte Carlo, utilizzando rispettivamente il processo lognormale standard e quello binario.
Si utilizzino i dati anagrafici e di mercato riportati nel punto (A). Confrontare e discutere i risultati ottenuti al variare di m, fornendone una adeguata spiegazione.

Caratteristiche del punto:

- **Propedeuticità: punti (A) e (E-1).**
- **Grado di difficoltà: Medio**

- **Tempo di sviluppo atteso: Basso**
- **Ai fini dell'esame: Facoltativo**

I risulti ottenuti devono essere presentati sotto forma di breve discussione scritta, con l'ausilio eventuale di grafici e tabelle.
La relazione deve contenere in appendice il listato integrale del codice.
La relazione (in formato .pdf o .doc) e l'intera libreria (file .hhp, .cpp e diagramma della libreria) vanno prodotti anche in formato elettronico (file .zip).

4.6 Esercizio serie a.6

4.6.1 Introduzione

Il presente capitolo è dedicato all'esposizione e discussione di una sessione di laboratorio / esercitazione relativa al metodo Monte Carlo presentato nel capitolo 2.1.
Nel primo paragrafo verrà presentato il contratto di una opzione esotica di cui ci proponiamo di calcolare il prezzo tramite una simulazione Monte Carlo. Nel par. 4.6.3 vengono presentate una serie di considerazioni. Infine nel par. 4.6.4 vengono esposti i punti su cui focalizzare l'esercitazione.
L'obiettivo della presente esercitazione non è solo quello di svolgere un semplice esercizio di programmazione, implementando una simulazione Monte Carlo, ma di mostrare piuttosto come l'integrazione dell'approccio numerico con considerazioni qualitative e analitiche è in grado di portare ad una comprensione completa del problema.

4.6.2 Definizione del problema

Si consideri un contratto opzionale con le seguenti caratteristiche:

- Opzione con esercizio europeo (ovvero il diritto può essere fatto valere unicamente alla data di scadenza);
- Sia T la data di maturità e $t = 0$ la data attuale. Il sottostante dell'opzione sia un'azione, il cui prezzo al tempo t venga indicato con $S(t)$;
- Supponiamo che il prezzo $S(t)$ dell'azione segua un processo log-normale, con μ costante e volatilità σ costante (ovvero l'usuale modello browniano utilizzato in finanza);
- Il pay-off pagato a scadenza sia:

$$\text{Pay-off}_{\text{Performance with corridor}} = 1 \text{ euro} \cdot \text{Max}\left[P - E, 0\right] \ , \tag{4.24}$$

  dove E è una sorta di strike price dell'opzione, P è la somma delle variabili P_i definite sotto. Queste sono legate all'andamento del sottostante sugli m intervalli equispaziati di ampiezza $\delta_t = T/m$, in cui è suddiviso l'intevallo temporale $(0, T)$, da oggi alla data di maturità dell'opzione. Ovvero:

$$P = \sum_{i=0}^{m-1} P_i \ , \tag{4.25}$$

$$P_i = \text{Min}\left[\text{Max}\left[S(t_{i+1})/S(t_i) - 1, l\right], u\right] \ , \tag{4.26}$$

dove:

$$l = \frac{e^{r\,\delta_t}}{k} - 1\ , \tag{4.27}$$

$$u = k\,e^{r\,\delta_t} - 1\ , \tag{4.28}$$

e $t_i = i\ T/m$ per $i = 0, ..., m$.
La costante k, presente nelle equazioni 4.27 e 4.28, viene parametrizzata tramite la seguente relazione:

$$k = 1 + \lambda\ \sqrt{\delta_t}\ , \tag{4.29}$$

che ne definisce la legge di scala rispetto a δ_t. λ indica una costante fissa, invariante rispetto a tutti i parametri di mercato coinvolti.
P_i rappresenta la performance dell'azione sull'intervallo i-esimo, limitata inferiormente e superiormente da l ed u. In altri termini le grandezze l e u rappresentano rispettivamente il valore minimo e massimo assunto dalla variabile P_i.
r rappresenta il tasso risk free (si supporrà quindi implicitamente che la curva dei tassi risk free sia piatta, ovvero che il tasso risk free non dipenda dalla scadenza considerata).
λ (e conseguentemente k) è un parametro in grado di restringere o allargare il corridoio (l, u) e soddisfa sempre il seguente vincolo:

$$\lambda >= 0\ . \tag{4.30}$$

Si noti che per $\lambda = 0$ (i.e. k=1) il corridoio si stringe completamente e P_i è vincolato ad assumere il valore: $\left(e^{r\,\delta_t} - 1\right)$; viceversa per $\lambda = \infty$ (i.e. $k = \infty$), P_i coincide con la variazione percentuale (performance) del prezzo del sottostante, relativamente all'intervallo i-esimo.

4.6.3 Alcune considerazioni

Il pay-off dell'opzione definita sopra è simile al classico pay-off di una call di tipo asiatico con l'importante differenza che la variabile P_i risulta più complessa di quanto non avvenga in un'ordinaria opzione asiatica. Inoltre per $m > 1$, il contratto diventa di tipo path-dependent, in quanto P è definito in base al percorso seguito dal sottostante durante la vita dell'opzione (in particolare dipende dai valori assunti dal sottostante in corrispondenza delle date di fixing $\{t_i\}$).
Si noti come nel caso limite $k = +\infty$ e $m = 1$, il pay-off sia riconducibile, dopo semplici manipolazini algebriche ad una call ordinaria.

4.6.4 Obiettivi dell'esercitazione

Alla fine di ogni punto viene riportato quanto segue:

- **Propedeuticità.**
- **Grado di difficoltà: (Basso, Medio, Alto, Molto Alto)**
- **Tempo di sviluppo atteso: (Breve, Medio, Lungo)**
- **Richiesta ai fini dell'esame: (Obbligatorio, Facoltativo)**

Si richiede di sviluppare e risolvere i seguenti punti:

(A) *Sviluppo libreria ad oggetti e pricing Monte Carlo*

(A-1) Sviluppare un programma ad oggetti in C++ che implementi una simulazione Monte Carlo per il pricing dell'opzione esotica descritta nel paragrafo 4.6.2. Tale programma deve prendere come input i seguenti dati:

```
Numero di simulazioni Monte Carlo
   N = 10.000
   N = 20.000
   N = 40.000

(in generale i valori riportati sopra sono solo indicativi. E' lasciata
allo studente la scelta di quali valori assumere per tale grandezza).

Valore dell'azione all'istante iniziale
   S_0 = 100

Valore dello strike price (in %)
   E = 1   %
   E = 1.5 %
   E = 2   %
   E = 3   %
   E = 4   %
   E = 5   %
   E = 6   %
   E = 7   %
   E = 8   %
   E = 9   %
   E = 10  %
   E = 20  %
   E = 50  %

(in generale i valori riportati sopra sono solo indicativi. E' lasciata
allo studente la scelta di quali valori assumere per tale grandezza).
```

```
Valore del tasso risk free
   r = 2 %

Valore della volatilita', su base annua, del sottostante
   sigma = 30 %

Durata dell'opzione T pari a:
   T = 0.5 anni
   T = 1.0 anni
   T = 1.5 anni
   T = 2.0 anni
   T = 2.5 anni
   T = 3.0 anni
   T = 3.5 anni
   T = 4.0 anni
   T = 10.0 anni

(in generale i valori riportati sopra sono solo indicativi. E' lasciata
allo studente la scelta di quali valori assumere per tale grandezza).


Numero di intervalli m pari a:
   m = 2
   m = 5
   m = 10
   m = 20
   m = 50
   m = 100
   m = 200

(in generale i valori riportati sopra sono solo indicativi. E' lasciata
allo studente la scelta di quali valori assumere per tale grandezza).

Valore di lambda (in %)
   lambda = 0   %
   lambda = 15  %
   lambda = 30  %
   lambda = 60  %
   lambda = 120  %
```

In particolare si implementi il metodo Monte Carlo descritto nel paragrafo 2.1.4 utilizzando: a) il metodo approssimato di Eulero

e b) la formula integrale esatta (2.18).

Caratteristiche del punto:

* **Propedeuticità: nessuna.**
* **Grado di difficoltà: Medio**
* **Tempo di sviluppo atteso: Lungo**
* **Ai fini dell'esame: obbligatorio**

(A-2) Produrre un insieme di grafici che mostrino l'andamento del prezzo dell'opzione per differenti valori di T, λ e m (in particolare si faccia riferimento ai valori di T, λ e m specificati nel punto A-1). Indicativamente il grafico riporterà in ascissa il tempo a maturità T (fissato λ) oppure λ (fissato T) e in ordinata il valore dell'opzione. Verranno riportati sul medesimo grafico le curve relative a differenti valori di m.

Caratteristiche del punto:

* **Propedeuticità: (A-1).**
* **Grado di difficoltà: Medio**
* **Tempo di sviluppo atteso: Lungo**
* **Ai fini dell'esame: obbligatorio**

(A-3) Produrre un insieme analogo di grafici (come nel punto A-2) che mostrino l'andamento dell'errore nella stima del prezzo dell'opzione per differenti valori di (T,λ,m), fissato $N = 10.000$.
N.B. L'errore va calcolato percentualmente ed in forma relativa, ovvero: errore MC assoluto / prezzo opzione, espresso in %.

Caratteristiche del punto:

* **Propedeuticità: (A-1).**
* **Grado di difficoltà: Medio**
* **Tempo di sviluppo atteso: Lungo**
* **Ai fini dell'esame: obbligatorio**

(B) *Formule di valutazione esatte*

(B-1) i) Ricavare una formula analitica esatta per l'opzione nel caso in cui $m = 1$ (con k e σ qualunque). ii) Confrontare i risultati ottenuti dalla formula esatta con le stime Monte Carlo per $m = 1$.

Caratteristiche del punto:

* **Propedeuticità: punto (A) per quanto riguarda il confronto con i dati Monte Carlo. Nessuna per quanto riguarda la derivazione della formula analitica.**
* **Grado di difficoltà: Medio/Alto**

- **Tempo di sviluppo atteso: Breve**
- **Ai fini dell'esame: Facoltativo**

(B-2) i) Ricavare una formula analitica esatta per l'opzione nel caso in cui $\lambda = 0$ (con m e σ qualunque). ii) Confrontare i risultati ottenuti dalla formula esatta con le stime Monte Carlo per $\lambda = 0$.

Caratteristiche del punto:

- **Propedeuticità: punto (A) per quanto riguarda il confronto con i dati Monte Carlo. Nessuna per quanto riguarda la derivazione della formula analitica.**
- **Grado di difficoltà: Basso**
- **Tempo di sviluppo atteso: Breve**
- **Ai fini dell'esame: Facoltativo**

(B-3) i) Ricavare una formula analitica esatta per l'opzione nel caso in cui $\sigma = 0$ (con m e λ qualunque).

Caratteristiche del punto:

- **Propedeuticità: nessuna. Può essere d'aiuto aver risolto il punto (B-2).**
- **Grado di difficoltà: Basso**
- **Tempo di sviluppo atteso: Breve**
- **Ai fini dell'esame: Facoltativo**

(C) *Formule di valutazione approssimate/asintotiche*

(C-1) i) Ricavare una formula analitica per calcolare il prezzo dell'opzione nel caso in cui $\delta_t \to 0$ e $k = \infty$ (i.e. $m \to \infty$ con T fissato, $\lambda \to \infty$ e $\lambda\,\delta_t \to \infty$). ii) Confrontare i risultati ottenuti dalla formula esatta con le stime Monte Carlo per $k \to \infty$.

Caratteristiche del punto:

- **Propedeuticità: punto (A) per quanto riguarda il confronto con i dati Monte Carlo. Nessuna per quanto riguarda la derivazione della formula analitica.**
- **Grado di difficoltà: Medio/Alto**
- **Tempo di sviluppo atteso: Breve**
- **Ai fini dell'esame: Facoltativo**

(C-2) Discutere il comportamento dei grafici e la loro interpretazione finanziaria quando il numero di intervalli m diventa molto grande (in linea di principio infinito). Fornire un'interpretazione teorica dei risultati ottenuti per m molto grande, in particolare:
i) derivare una formula analitica asintotica per il calcolo del prezzo dell'opzione nel limite $m \to \infty$. Si supponga che la costante k scali con δ_t tramite la relazione 4.30 (ovvero si consideri λ costante

e conseguentemente $k \to 0$ nel limite in cui $\delta_t \to 0$).
ii) si confrontino i risultati ottenuti dalla formula asintotica con le stime Monte Carlo.

Caratteristiche del punto:

* **Propedeuticità: punto (A) per quanto riguarda il confronto con i dati Monte Carlo. Per la derivazione della formula analitica asintotica, si suggerisce di risolvere prima i punti (B-2) e (C-1).**
* **Grado di difficoltà: Molto alto**
* **Tempo di sviluppo atteso: Medio**
* **Ai fini dell'esame: Facoltativo**

(D) *Tecniche di riduzione dell'errore*

(D-1) Allo scopo di ridurre gli errori statistici, si utilizzi: la tecnica della variabile antitetica [11]. In particolare si richiede di:
i) valutare il prezzo dell'opzione ed il relativo errore utilizzando la tecnica della variabile antitetica;
ii) calcolare e discutere la riduzione degli errori statistici (ovvero le deviazioni standard sui pay-off) ottenuta tramite l'uso della variabile antitetica.

Caratteristiche del punto:

* **Propedeuticità: (A).**
* **Grado di difficoltà: Basso**
* **Tempo di sviluppo atteso: Medio**
* **Ai fini dell'esame: Obbligatorio**

(D-2) Rispetto al metodo standard della variabile di controllo [12], in cui:

$$x_{\text{MC new}} = x_{\text{MC}} + y - y_{\text{MC}} \ , \tag{4.31}$$

si sviluppi una variante della formula sopra riportata al fine di sfruttare pienamente l'informazione contenuta nella variabile di controllo.

Caratteristiche del punto:

* **Propedeuticità: Nessuna.**
* **Grado di difficoltà: Medio/Alto**
* **Tempo di sviluppo atteso: Medio**

[11] Vedi par. 2.1.4.e
[12] Vedi par. 2.1.4.e

- **Ai fini dell'esame: Facoltativo**

(D-3) Sfruttando il risultato ottenuto nel punto (B1), allo scopo di ridurre gli errori statistici, si ricorra alla tecnica della variabile di controllo, nella sua versione migliorata, derivata nel punto precedente. In particolare si richiede di:
i) di valutare il prezzo dell'opzione ed il relativo errore utilizzando la variabile di controllo.
ii) calcolare e discutere la riduzione degli errori statistici (ovvero le deviazioni standard sui pay-off) ottenuta tramite l'uso della variabile di controllo al variare di m.

Caratteristiche del punto:

- **Propedeuticità: (B-1) e (D-2).**
- **Grado di difficoltà: Medio**
- **Tempo di sviluppo atteso: Medio**
- **Ai fini dell'esame: Facoltativo**

(E) *Processi stocastici*

(E-1) Sviluppare una classe per gestire un processo alternativo a quello lognormale standard, in particolare tale processo è definito sostituendo nella formula (1.36), al posto della variabile gaussiana w a media nulla e varianza unitaria, una variabile stocastica alternativa z (sempre a media nulla e varianza unitaria), definita come segue:

$$z = \begin{cases} +1 & \text{con probabilità } 0.5 \\ -1 & \text{con probabilità } 0.5 \end{cases} \tag{4.32}$$

Il prezzo dell'azione è definito da:

$$S(\delta_t) = S(0)\, e^{\left(r - \frac{\sigma^2}{2}\right)\delta_t + \sigma\sqrt{\delta_t}\, z}\ , \tag{4.33}$$

Il processo in questione ha natura binaria, nel senso che consente unicamente un movimento al ribasso o al rialzo dell'azione su un certo intervallo di tempo δ_t.

Si verifichi che la libreria sia strutturata in maniera tale da consentire l'utilizzo del nuovo processo senza apportare modifiche alla libreria (a parte naturalmente l'aggiunta e la definizione della nuova classe per la descrizione del processo binario). Si discuta quali siano i vantaggi di un tale approccio e se questi siano o meno ottenibili nell'ambito di una programmazione procedurale.

Caratteristiche del punto:

- **Propedeuticità: punto (A).**

- **Grado di difficoltà: Medio**
- **Tempo di sviluppo atteso: Medio**
- **Ai fini dell'esame: Facoltativo**

(E-2) Si consideri il pricing di un'opzione plain vanilla, spezzando l'intervallo $(0, T)$ (T essendo la data di maturità dell'opzione) in m intervalli. Effettuare la valutazione del prezzo dell'opzione plain vanilla con metodo Monte Carlo, utilizzando rispettivamente il processo lognormale standard e quello binario.
Si utilizzino i dati anagrafici e di mercato riportati nel punto (A). Confrontare e discutere i risultati ottenuti al variare di m, fornendone una adeguata spiegazione.

Caratteristiche del punto:

- **Propedeuticità: punti (A) e (E-1).**
- **Grado di difficoltà: Medio**
- **Tempo di sviluppo atteso: Medio**
- **Ai fini dell'esame: Facoltativo**

(F) *Programmazione in C++ di tipo avanzato*

(F-1) Si utilizzi il design pattern singleton [13] all'interno della libreria, per risolvere una serie di tematiche quali:

- gestione di code di stampa o code degli errori;
- garantire l'unicità del generatore di numeri casuali, in modo che l'istanziazione di più oggetti di tipo Process utilizzi sempre, per costruzione, lo stesso generatore di numeri pseudo casuali.

Caratteristiche del punto:

- **Propedeuticità: punti (A).**
- **Grado di difficoltà: Medio**
- **Tempo di sviluppo atteso: Medio**
- **Ai fini dell'esame: Facoltativo**

(F-2) definire delle classi per gestire il concetto di tempo (classe Time) e di intervallo di tempo (classe Time_interval).

Caratteristiche del punto:

- **Propedeuticità: punti (A).**
- **Grado di difficoltà: Medio**
- **Tempo di sviluppo atteso: Alto**
- **Ai fini dell'esame: Facoltativo**

[13] Si veda ad es. la seguente pagina web per una descrizion del design pattern singleton: http://it.wikipedia.org/wiki/Singleton

(F-3) Una volta implementato il concetto di classe Time, utilizzare il singleton (o più propriamente una sua variazione) in modo da avere un unico oggetto di tipo Time che rappresenti l'istante attuale. In altri termini si richiede di definire un metodo di Time (Time::Get_present_time()) che resituisca sempre il pointer ad un unico oggetto contenente il tempo attuale.

Caratteristiche del punto:

* **Propedeuticità: punti (A) e (F-2).**
* **Grado di difficoltà: Medio**
* **Tempo di sviluppo atteso: Medio**
* **Ai fini dell'esame: Facoltativo**

(F-4) utilizzare il concetto di polimorfismo in modo avanzato, con l'obiettivo di rendere la libreria facilmente estensibile e flessibile (ad esempio consentendo di trattare processi stocastici differenti e random number diversi da un semplice double; si veda a tal proposito il punto E-1).

Caratteristiche del punto:

* **Propedeuticità: punti (A).**
* **Grado di difficoltà: Alto**
* **Tempo di sviluppo atteso: Alto**
* **Ai fini dell'esame: Facoltativo**

I risulti ottenuti devono essere presentati sotto forma di breve discussione scritta, con l'ausilio eventuale di grafici e tabelle.
La relazione deve contenere in appendice il listato integrale del codice.
La relazione (in formato .pdf o .doc) e l'intera libreria (file .hhp, .cpp e diagramma della libreria) vanno prodotti anche in formato elettronico (file .zip).

4.7 Esercizio serie a.7

4.7.1 Introduzione

Il presente capitolo è dedicato all'esposizione e discussione di una sessione di laboratorio / esercitazione relativa al metodo Monte Carlo presentato nel capitolo 2.1.
Nel primo paragrafo verrà presentato il contratto di una opzione di cui ci proponiamo di calcolare il prezzo tramite una simulazione Monte Carlo. Nel par. 4.7.3 vengono presentate una serie di considerazioni. Infine nel par. 4.7.4 vengono esposti i punti su cui focalizzare l'esercitazione.
L'obiettivo della presente esercitazione non è solo quello di svolgere un semplice esercizio di programmazione, implementando una simulazione Monte Carlo, ma di mostrare piuttosto come l'integrazione dell'approccio numerico con considerazioni qualitative e analitiche sia in grado di portare ad una comprensione completa del problema.

4.7.2 Definizione del problema

Si consideri un contratto opzionale con le seguenti caratteristiche:

- Opzione con esercizio europeo (ovvero il diritto può essere fatto valere unicamente alla data di scadenza);
- Sia T la data di maturità e $t = 0$ la data attuale. Il sottostante dell'opzione sia un'azione, il cui prezzo al tempo t venga indicato con $S(t)$;
- Supponiamo che il prezzo $S(t)$ dell'azione segua un processo log-normale, con μ costante e volatilità σ costante (ovvero l'usuale modello browniano utilizzato in finanza);
- Il pay-off pagato a scadenza sia:

$$\text{Pay-off} = \begin{cases} E_2 - E_1 & \text{se } S(T) > E_2 \ , \\ S(T) - E_1 & \text{se } E_1 < S(T) < E_2 \ , \\ 0 & \text{se } S(T) < E_1 \ . \end{cases} \tag{4.34}$$

  dove E_1 and E_2 sono i due "strike price" dell'opzione, con $E_2 > E_1$.

4.7.3 Alcune considerazioni

Il pay-off dell'opzione definita sopra è apparentemente più complesso di quello di una semplice plain vanilla call.
Il contratto non è path-dependent, in quanto il pay off dipende solo dal valore assunto dal sottostante a scadenza e non dal percorso seguito.

4.7.4 Obiettivi dell'esercitazione

Alla fine di ogni punto viene riportato quanto segue:

- **Propedeuticità.**
- **Grado di difficoltà: (Basso, Medio, Alto, Molto Alto)**
- **Tempo di sviluppo atteso: (Breve, Medio, Lungo)**
- **Richiesta ai fini dell'esame: (Obbligatorio, Facoltativo)**

Si richiede di sviluppare e risolvere i seguenti punti:

(A) *Sviluppo libreria ad oggetti e pricing Monte Carlo*

(A-1) Sviluppare un programma ad oggetti in C++ che implementi una simulazione Monte Carlo per il pricing dell'opzione descritta nel paragrafo 4.7.2. Tale programma deve prendere come input i seguenti dati:

```
Numero di simulazioni Monte Carlo
   N = 10.000
   N = 20.000
   N = 40.000

(in generale i valori riportati sopra sono solo indicativi. E' lasciata
allo studente la scelta di quali valori assumere per tale grandezza).

Valore dell'azione all'istante iniziale
   S_0 = 100

Valore dello strike price (in %)
   E1 = 105
   E2 = 110

(in generale i valori riportati sopra sono solo indicativi. E' lasciata
allo studente la scelta di quali valori assumere per tale grandezza).

Valore del tasso risk free
   r = 2 %

Valore della volatilita', su base annua, del sottostante
   sigma = 30 %

Durata dell'opzione T pari a:
```

```
T = 0.5 anni
T = 1.0 anni
T = 1.5 anni
T = 2.0 anni
T = 2.5 anni
T = 3.0 anni
T = 3.5 anni
T = 4.0 anni
T = 10.0 anni
```

`(in generale i valori riportati sopra sono solo indicativi. E' lasciata allo studente la scelta di quali valori assumere per tale grandezza).`

In particolare si implementi il metodo Monte Carlo descritto nel paragrafo 2.1.4 utilizzando: a) il metodo approssimato di Eulero e b) la formula integrale esatta (2.18).
Nel caso si utilizzi la formula approssimata di Eulero sarà necessario suddividere l'intervallo $(0, T)$ in m intervalli equispaziati di ampiezza $\delta t = T/m$. E' lasciata allo studente la scelta di quali valori assumere per m.
Si risponda poi alle seguenti domande:

(i) Nel caso si utilizzi l'approssimazione di Eulero per generare i cammini, può accadere che il prezzo simulato dell'azione sia negativo?

(ii) Quali strategie si possono adottare per evitare che ciò avvenga?

(iii) Detta δt la lunghezza temporale del singolo step Monte Carlo, è più probabile che tale problema insorga con δt piccolo oppure grande? Per quale motivo?

(iv) In quale limite i risultati ottenuti con l'approssimazione di Eulero convergono a quelli ottenuti con la formula integrale esatta?

Caratteristiche del punto:

* **Propedeuticità: nessuna.**
* **Grado di difficoltà: Medio**
* **Tempo di sviluppo atteso: Lungo**
* **Ai fini dell'esame: obbligatorio**

(A-2) Produrre un insieme di grafici che mostrino l'andamento del prezzo dell'opzione per differenti valori di T (in particolare si faccia riferimento ai valori di T specificati nel punto A-1). Indicativamente il grafico riporterà in ascissa il tempo a maturità T e

in ordinata il valore dell'opzione. Verranno riportati sul medesimo grafico le curve relative a differenti valori di m per quanto riguarda i dati ottenuti sotto l'approssimazione di Eulero.

Caratteristiche del punto:

* **Propedeuticità: (A-1).**
* **Grado di difficoltà: Medio**
* **Tempo di sviluppo atteso: Medio**
* **Ai fini dell'esame: obbligatorio**

(A-3) Produrre un insieme analogo di grafici (come nel punto A-2) che mostrino l'andamento dell'errore nella stima del prezzo dell'opzione per differenti valori di T, fissato $N = 10.000$.
N.B. L'errore va calcolato percentualmente ed in forma relativa, ovvero: errore MC assoluto / prezzo opzione, espresso in %.

Caratteristiche del punto:

* **Propedeuticità: (A-1).**
* **Grado di difficoltà: Medio**
* **Tempo di sviluppo atteso: Medio**
* **Ai fini dell'esame: obbligatorio**

(B) *Formule di valutazione esatte*

(B-1) i) Ricavare una formula analitica esatta. ii) Confrontare i risultati ottenuti dalla formula esatta con le stime Monte Carlo.

Caratteristiche del punto:

* **Propedeuticità: punto (A) per quanto riguarda il confronto con i dati Monte Carlo. Nessuna per quanto riguarda la derivazione della formula analitica.**
* **Grado di difficoltà: Medio/Alto**
* **Tempo di sviluppo atteso: Medio/breve**
* **Ai fini dell'esame: Facoltativo**

(C) *Tecniche di riduzione dell'errore*

(C-1) Allo scopo di ridurre gli errori statistici, si utilizzi: la tecnica della variabile antitetica [14]. In particolare si richiede di:
i) valutare il prezzo dell'opzione ed il relativo errore utilizzando la tecnica della variabile antitetica;
ii) calcolare e discutere la riduzione degli errori statistici (ovvero le deviazioni standard sui pay-off) ottenuta tramite l'uso della variabile antitetica.

Caratteristiche del punto:

[14] Vedi par. 2.1.4.e

* **Propedeuticità: (A).**
* **Grado di difficoltà: Basso**
* **Tempo di sviluppo atteso: Medio**
* **Ai fini dell'esame: Facoltativo**

(D) *Processi stocastici*

(D-1) Sviluppare una classe per gestire un processo alternativo a quello lognormale standard, in particolare tale processo è definito sostituendo nella formula (1.36), al posto della variabile gaussiana w a media nulla e varianza unitaria, una variabile stocastica alternativa z (sempre a media nulla e varianza unitaria), definita come segue:

$$z = \begin{cases} +1 & \text{con probabilità } 0.5 \\ -1 & \text{con probabilità } 0.5 \end{cases} \tag{4.35}$$

Il prezzo dell'azione è definito da:

$$S(\delta_t) = S(0)\, e^{\left(r - \frac{\sigma^2}{2}\right)\delta_t + \sigma\sqrt{\delta_t}\, z}\,, \tag{4.36}$$

Il processo in questione ha natura binaria, nel senso che consente unicamente un movimento al ribasso o al rialzo dell'azione su un certo intervallo di tempo δt.

Si verifichi che la libreria sia strutturata in maniera tale da consentire l'utilizzo del nuovo processo senza apportare modifiche alla libreria (a parte naturalmente l'aggiunta e la definizione della nuova classe per la descrizione del processo binario). Si discuta quali siano i vantaggi di un tale approccio e se questi siano o meno ottenibili nell'ambito di una programmazione procedurale.

Caratteristiche del punto:

* **Propedeuticità: punto (A).**
* **Grado di difficoltà: Medio**
* **Tempo di sviluppo atteso: Medio**
* **Ai fini dell'esame: Facoltativo**

(D-2) Si consideri il pricing di un'opzione plain vanilla, spezzando l'intervallo $(0, T)$ (T essendo la data di maturità dell'opzione) in m intervalli. Effettuare la valutazione del prezzo dell'opzione plain vanilla con metodo Monte Carlo, utilizzando rispettivamente il processo lognormale standard e quello binario.
Si utilizzino i dati anagrafici e di mercato riportati nel punto (A). Confrontare e discutere i risultati ottenuti al variare di m, fornendone una adeguata interpretazione.

Caratteristiche del punto:

- **Propedeuticità: punti (A) e (D-1).**
- **Grado di difficoltà: Medio**
- **Tempo di sviluppo atteso: Medio**
- **Ai fini dell'esame: Facoltativo**

(E) *Programmazione in C++ di tipo avanzato*

(E-1) Si utilizzi il design pattern singleton [15] all'interno della libreria, per risolvere una serie di tematiche quali:

- gestione di un generatore di numeri casuali, code di stampa o code degli errori;
- garantire l'unicità del generatore di numeri casuali, in modo che l'istanziazione di più oggetti di tipo Process utilizzi sempre, per costruzione, lo stesso generatore di numeri pseudo casuali.

Caratteristiche del punto:

- **Propedeuticità: punti (A).**
- **Grado di difficoltà: Medio**
- **Tempo di sviluppo atteso: Medio**
- **Ai fini dell'esame: Facoltativo**

(E-2) utilizzare il concetto di polimorfismo in modo avanzato, con l'obiettivo di rendere la libreria facilmente estensibile e flessibile (ad esempio consentendo di trattare processi stocastici differenti; si veda a tal proposito il punto D-1).

Caratteristiche del punto:

- **Propedeuticità: punti (A).**
- **Grado di difficoltà: Alto**
- **Tempo di sviluppo atteso: Alto**
- **Ai fini dell'esame: Facoltativo**

(E-3) modificare la classe Yield_curve in modo da gestire una struttura a termine per il tasso (ovvero una curva tasso che non sia piatta).

Caratteristiche del punto:

- **Propedeuticità: punti (A).**
- **Grado di difficoltà: Medio**
- **Tempo di sviluppo atteso: Medio**
- **Ai fini dell'esame: Facoltativo**

[15] Si veda ad es. la seguente pagina web per una descrizion del design pattern singleton: http://it.wikipedia.org/wiki/Singleton

I risulti ottenuti devono essere presentati sotto forma di breve discussione scritta, con l'ausilio eventuale di grafici e tabelle.
La relazione deve contenere in appendice il listato integrale del codice.
La relazione (in formato .pdf o .doc) e l'intera libreria (file .hhp, .cpp e diagramma della libreria) vanno prodotti anche in formato elettronico (file .zip).

4.8 Esercizio Serie b.1

4.8.1 Introduzione

Il presente capitolo è dedicato all'esposizione e discussione di una sessione di laboratorio / esercitazione relativa al metodo Monte Carlo presentato nelle note in allegato.
Nel primo paragrafo verranno descritti alcuni pay-off di opzioni esotiche di cui si richiederà una stima del prezzo tramite simulazione Monte Carlo. Nel par. 4.8.3 vengono esposti i punti su cui focalizzare l'esercitazione.

4.8.2 Definizione del problema

Si considerino dei contratti opzionali con le seguenti caratteristiche:

- Opzione plain vanilla call o put:

$$\begin{aligned} \text{Pay-off}_{\text{call plain vanilla}} &= Max\left[S(T) - E, 0\right] \\ \text{Pay-off}_{\text{put plain vanilla}} &= Max\left[E - S(T), 0\right] \ , \end{aligned} \tag{4.37}$$

  dove T rappresenta la data di maturità ed E lo strike price dell'opzione.

- Opzione con barriera down & out:

$$\begin{aligned} \text{Pay-off}_{\text{call down \& out}} &= I \cdot Max\left[S(T) - E, 0\right] \\ \text{Pay-off}_{\text{put down \& out}} &= I \cdot Max\left[E - S(T), 0\right] \ , \end{aligned} \tag{4.38}$$

  dove:

$$I = \begin{cases} 1 & \text{se } S(t_i) > B \quad \forall t_i = \delta t \cdot i \\ 0 & \text{diversamente} \end{cases} \tag{4.39}$$

  essendo B il valore della barriera.
  In altri termini il pay-off è quello di una plain vanilla con il vincolo seguente: se durante la vita dell'opzione, in corrispondenza delle date di fixing, il sottostante tocca la barriera, l'opzione si considera cancellata senza pagamento di alcun pay-off.

- Opzione reverse cliquet:

$$\begin{aligned} \text{Pay-off}_{\text{reverse call cliquet}} &= Max\left\{L, H - \sum_{i=0}^{N-1} Max\left[\frac{S(t_{i+1})}{S(t_i)} - 1, 0\right]\right\} \\ \text{Pay-off}_{\text{reverse put cliquet}} &= Max\left\{L, H - \sum_{i=0}^{N-1} Max\left[1 - \frac{S(t_{i+1})}{S(t_i)}, 0\right]\right\} \end{aligned} \tag{4.40}$$

  essendo L ed H rispettivamente il valore minimo e massimo del pay-off.

4.8.3 Obiettivi dell'esercitazione

Si richiede di:

(A) Implementare per ciascun pay-off, elencato nel paragrafo precedente, l'algoritmo Monte Carlo standard, tramite uno o più dei seguenti ambienti di sviluppo: (i) excel; (ii) codice VBA in excel; (iii) matlab e/o (iv) C++ ad oggetti (quest'ultimo facoltativo e solo per chi è fortemente interessato a tale approccio).
Relativamente all'implementazione del metodo Monte Carlo descritto nel paragrafo 2.1.4, si utilizzi in un primo momento la discretizzazione di Eulero del processo stocastico definita dall'equazione (2.14) e successivamente si ricorra alla formulazione esatta (2.18).
Si osservi che l'utilizzo di excel nell'implementazione del Monte Carlo ha finalità puramente didattiche, essendo inadatto a gestire un algoritmo "cpu intensive".
Si utilizzino i seguenti dati di mercato ed i seguenti settings:

```
# Numero di simulazioni Monte Carlo
N = 100, 400, 1600 (valori superiori solo nel caso si
    utilizzi VBA o matlab)

# Valore dell'azione all'istante iniziale
S_0 = 100

# Valore del tasso risk free
r = 5 %

# Valore della volatilita', su base annua, del sottostante
sigma = 10%, 15%, 25% e 30%

# Per l'opzione plain vanilla e con barriera, strike price:
#
E = 100

# Per l'opzione con barriera
#
B = 95

# Per l'opzione reverse cliquet
#
L = 2 %
H = 12 %

# Data di maturit\'a dell'opzione (espressa in anni)
```

```
T = 1y, 8y, 20y

# Numero di intervalli
num_interavalli = 12
```

N.B.: fatta eccezione per i punti (A) e (B), si faccia sempre riferimento alla formula (2.18) per la generazione dei cammini aleatori.

(B) Nel caso si utilizzi l'approssimazione di Eulero per generare i cammini, può accadere che il prezzo simulato dell'azione sia negativo? Quali strategie si possono adottare per evitare che ciò avvenga? Detta δt la lunghezza temporale del singolo step Monte Carlo, è più probabile che tale problema insorga quando δt è piccolo oppure grande? Per quale motivo?

(C) Calcolare l'andamento dell'errore Monte Carlo per diversi valori di N (da 1000 fino a 2000 con intervalli di 500). Verificare che questo scali con N secondo la legge:

$$\epsilon = \frac{K(\sigma)}{\sqrt{N}} \tag{4.41}$$

dove $K(\sigma)$ rappresenta una semplice costante moltiplicativa (dipendente in generale dai dati di mercato, dal tipo di pay-off, ecc.). Nel caso dell'equazione sopra, è stata evidenziata la sola dipendeza di K da σ.

(D) Come indicato, l'andamento dell'errore Monte Carlo scala con N tramite la legge (4.41). L'andamento con $1/\sqrt{N}$ è tipico ed intrinseco del Monte Carlo, viceversa la costante al numeratore, $K(\sigma)$, dipende: dal problema (ovvero dal pay-off), dai dati di mercato (tra cui ad esempio la volatilità) e dalla metodologia con cui il Monte Carlo viene implementato (ad esempio nel caso si utilizzino metodi di riduzione delle varianza. Si veda a tal proposito il punto successivo).
Determinare l'andamento di $K(\sigma)$ al variare della volatilità per i tre pay-off descritti nel paragrafo precedente. Cosa ci si deve attendere nel caso in cui $\sigma \to 0$? Quale intepretazione si può dare di questo comportamento?

(E) Al fine di ridurre gli errori statistici si utilizzi la tecnica della variabile antitetica (vedi par. 2.1.4.e).
Rispetto all'andamento dell'errore definito dall'eq(4.41), qual'e' l'impatto della variabile antitetica nella riduzione dell'errore? Ovvero: (a) cambia la legge di scala $1/\sqrt{N}$? (b) Cambia il valore di $K(\sigma)$ e come? In generale si ripeta l'analisi dei punti (C) e (D) e la si confronti con i risultati precedentemente ottenuti.

(F) Relativamente al caso del pay-off di un'opzione con barriera down & out, si calcoli tramite il metodo Monte Carlo la probabilità di tocco della barriera. Si valuti anche l'errore di tale stima.
Suggerimento: si valuti il valore atteso di I (definito dall'equazione 4.39).

(G) Cosa si può dire sul posizionamento reciproco del prezzo di una plain vanilla rispetto alla corrispondente versione con barriera down & out? In altri termini il prezzo della plain vanilla è maggiore o minore della versione con barriera? E per quali motivi?

I risulti ottenuti devono essere presentati sotto forma di breve discussione scritta, con l'ausilio eventuale di tabelle e grafici (eventualmente contenuti in un foglio excel se questo fosse ritenuto più comodo).

4.9 Esercizio serie b.2

4.9.1 Introduzione

Il presente capitolo è dedicato all'esposizione e discussione di una sessione di laboratorio / esercitazione / tesina relativa al metodo Monte Carlo.
Nel primo paragrafo verranno descritti alcuni pay-off di opzioni esotiche di cui si richiederà una stima del prezzo tramite simulazione Monte Carlo.
Nel paragrafo 4.9.3 verranno esposti una serie di modelli stocastici che saranno eventualmente oggetto delle domande del tema d'esame.
Nel par. 4.9.6 sono riportate tutte le domande del tema d'esame per tutti i gruppi. Ovviamente ciascun gruppo dovrà rispondere solo ad un sottoinsieme di queste domande (si veda nello specifico il par 4.9.7).
Infine nel par. 4.9.7 sono riportate le domande specifiche a cui il singolo gruppo è chiamato a rispondere. Per ciascun gruppo viene anche allegato il data set, ovvero il set dei dati di mercato e dei dettagli anagrafici dell'opzione, da utilizzare in alcune richieste del tema d'esame.

4.9.2 Definizione dei pay-off

Si considerino le seguenti opzioni:

- Opzione plain vanilla call o put:

$$\begin{aligned} \text{Pay-off}_{\text{call plain vanilla}} &= Max\left[S(T) - E, 0\right] \\ \text{Pay-off}_{\text{put plain vanilla}} &= Max\left[E - S(T), 0\right] \ , \end{aligned} \tag{4.42}$$

  dove T rappresenta la data di maturità, E lo strike price dell'opzione e $S(T)$ è il prezzo del sottostante a scadenza.

- Opzione digital call:

$$\text{Pay-off}_{\text{digital call}} = \begin{cases} 1 & \text{se } S(T) > E \\ 0 & \text{diversamente} \end{cases} \tag{4.43}$$

  dove T rappresenta la data di maturità, E lo strike price dell'opzione e $S(T)$ è il prezzo del sottostante a scadenza.

- Opzione con barriera down & out su singolo sottostante:

$$\begin{aligned} \text{Pay-off}_{\text{call down \& out}} &= I \cdot Max\left[S(T) - E, 0\right] \\ \text{Pay-off}_{\text{put down \& out}} &= I \cdot Max\left[E - S(T), 0\right] \ , \end{aligned} \tag{4.44}$$

  dove:

$$I = \begin{cases} 1 & \text{se } S(t_i) > B \quad \forall t_i = \delta t \cdot i \\ 0 & \text{diversamente} \end{cases} \tag{4.45}$$

essendo B il valore della barriera.
In altri termini il pay-off è quello di una plain vanilla con il vincolo seguente: se durante la vita dell'opzione, in corrispondenza ad almeno una delle date di fixing, il sottostante tocca la barriera, l'opzione si considera cancellata senza pagamento di alcun pay-off.

- Opzione digitale con barriera down & out su n sottostanti multipli:

$$\text{Pay-off}_{\text{digital down \& out}} = \begin{cases} K & \text{se } \frac{S_j(t_i)}{S_j(t_0)} > B \quad \forall t_i = \delta t \cdot i \,, \forall j = 1, \cdots n \\ 0 & \text{diversamente} \end{cases} \tag{4.46}$$

essendo K una costante (ad es. uguale ad 1 euro) e B il valore della barriera espresso questa volta in forma percentuale.
In altri termini il pay-off è pari ad un ammontare K condizionato al seguente vincolo: se durante la vita dell'opzione, in corrispondenza di almeno una delle date di fixing, almeno uno tra gli n sottostanti tocca la barriera, l'opzione si considera cancellata senza pagamento di alcun pay-off.
Si osservi che la barriera si considera toccata per il sottostante S_j al tempo t_i se l'incremento relativo dell'azione al tempo t_i rispetto al tempo t_0 (ovvero $S_j(t_i)/S_j(t_0)$) è inferiore a B (dove B è espresso appunto in forma percentuale).

- Opzione digitale con barriera su singolo sottostante:

$$\text{Pay-off}_{\text{digital barrier}} = \begin{cases} K & \text{se } S(t_i) > B \quad \forall t_i = \delta t \cdot i \\ 0 & \text{diversamente} \end{cases} \tag{4.47}$$

essendo K una costante (ad es. uguale a 1 euro) e B il valore della barriera.
In altri termini il pay-off è pari ad un ammontare K condizionato al fatto che per tutte le date di fixing il sottostante si mantenga sopra la barriera. Viceversa l'opzione si considera cancellata senza pagamento di alcun pay-off.

- Opzione digitale con doppia barriera (detta anche corridor) su singolo sottostante:

$$\text{Pay-off}_{\text{digital corridor}} = \begin{cases} K & \text{se } B_l < S(t_i) < B_u \quad \forall t_i = \delta t \cdot i \\ 0 & \text{diversamente} \end{cases} \tag{4.48}$$

essendo K una costante (ad es. uguale a 1 euro), B_l il valore della barriera inferiore e B_u il valore della barriera superiore. (Ovviamente $B_u > B_l$).
In altri termini il pay-off è pari ad un ammontare K condizionato al

fatto che per tutte le date di fixing il sottostante sia racchiuso all'interno del corridoio rappresentato dalla barriera inferiore e da quella superiore. Viceversa l'opzione si considera cancellata senza pagamento di alcun pay-off.

- Opzione reverse cliquet:

$$\begin{aligned}\text{Pay-off}_{\text{reverse call cliquet}} &= Max\left\{L, H - \sum_{i=0}^{N-1} Max\left[\frac{S(t_{i+1})}{S(t_i)} - 1, 0\right]\right\} \\ \text{Pay-off}_{\text{reverse put cliquet}} &= Max\left\{L, H - \sum_{i=0}^{N-1} Max\left[1 - \frac{S(t_{i+1})}{S(t_i)}, 0\right]\right\}\end{aligned} \tag{4.49}$$

essendo L ed H due costanti che rappresentano rispettivamente il valore minimo e massimo del pay-off.

- Opzione best of:

$$\begin{aligned}\text{Pay-off}_{\text{best of call}} &= Max\left\{Max\left[\frac{S_1(T)}{S_1(t_0)}, \frac{S_2(T)}{S_2(t_0)}\right] - E, 0\right\} \\ \text{Pay-off}_{\text{best of call}} &= Max\left\{E - Max\left[\frac{S_1(T)}{S_1(t_0)}, \frac{S_2(T)}{S_2(t_0)}\right], 0\right\}\end{aligned} \tag{4.50}$$

essendo E lo strike price espresso in forma percentuale.
Si osservi che un'opzione *best of* coinvolge due sottostanti S_1 e S_2.

4.9.3 Definizione dei modelli

Si considerino i seguenti modelli stocastici per descrivere l'evoluzione di un'azione (o a seconda dei casi di più azioni):

- Modello lognormale standard:

$$\frac{dS}{S} = r\,dt + \sigma\,\sqrt{dt}\,w \tag{4.51}$$

- Modello lognormale standard bivariato (due sottostanti):

$$\begin{aligned} \frac{dS_1}{S_1} &= r\,dt + \sigma_1\,\sqrt{dt}\,w_1 \\ \frac{dS_2}{S_2} &= r\,dt + \sigma_2\,\sqrt{dt}\,w_2 \\ \langle w_1 w_2 \rangle &= \rho \end{aligned} \tag{4.52}$$

- Modello lognormale standard per n sottostanti scorrelati:

$$\begin{aligned} \frac{dS_j}{S_j} &= r\,dt + \sigma_j\,\sqrt{dt}\,w_j \\ \langle w_{j_1} w_{j_2} \rangle &= 0 \;\; \forall j_1 \neq j_2 \end{aligned} \tag{4.53}$$

- Modello a salto

$$\frac{dS}{S} = \mu dt + \sigma\sqrt{dt}w + (J-1)dN\ , \tag{4.54}$$

  dove J è l'ampiezza del salto. L'intensità del salto è data da λ.
  Ricordiamo che la variabile stocastica dN può assumere solo i seguenti valori:

$$dN = \begin{cases} 1 & \text{con probabilità } \lambda\,dt \\ 0 & \text{con probabilità } 1 - \lambda\,dt \end{cases} \tag{4.55}$$

- Modello a salto con ampiezza variabile:

$$\frac{dS}{S} = \mu dt + \sigma\sqrt{dt}w + (J-1)dN\ , \tag{4.56}$$

  nel caso avvenga un salto ($dN = 1$) l'ampiezza di tale salto, J, non è deterministica ma è anch'essa casuale, determinata secondo la seguente legge:

$$J = \begin{cases} J_1 & \text{con probabilità } p \\ J_2 & \text{con probabilità } 1 - p \end{cases} \tag{4.57}$$

  L'intesità di salto è sempre espressa dalla costante λ.

- Modello a doppio salto:

$$\frac{dS}{S} = \mu dt + \sigma\sqrt{dt}w + (J_a - 1)dN_a + (J_b - 1)dN_b \,, \tag{4.58}$$

dove in ogni intervallo dt possono verificarsi due differenti tipologie di salti rappresentati rispettivamente da J_a e J_b, ciascuno con la propria intensità λ_a e λ_b. Ovvero:

$$dN_a = \begin{cases} 1 & \text{con probabilità } \lambda_a\, dt \\ 0 & \text{con probabilità } 1 - \lambda_a\, dt \end{cases} \tag{4.59}$$

e

$$dN_b = \begin{cases} 1 & \text{con probabilità } \lambda_b\, dt \\ 0 & \text{con probabilità } 1 - \lambda_b\, dt \end{cases} \tag{4.60}$$

4.9.4 Definizione dei metodi numerici utilizzabili

Il metodo Monte Carlo può essere implementato seguendo due schemi principali:

- schema di Eulero: tale schema si ottiene dalla discretizzazione dell'equazione stocastica. Per tale motivo è uno schema approssimato valido nel limite in cui $\Delta t \to 0$.
- schema esatto: tale schema si ottiene risolvendo esattamente in forma integrale l'equazione differenziale stocastica. Per tale ragione questo schema risulta esatto e può essere applicato a qualunque intervallo Δt grande a priori.

4.9.5 Tipi di implementazione

L'implementazione di quanto richiesto dal tema d'esame può essere effettuata tramite almeno uno dei seguenti ambienti di sviluppo/programmazione: (i) excel; (ii) codice VBA in excel; (iii) matlab o (iv) linguaggio ad oggetti C++ .

Va osservato che l'utilizzo di excel nell'implementazione del metodo Monte Carlo ha finalità puramente didattiche, essendo inadatto a gestire un algoritmo "cpu intensive". Per tale motivo l'implementazione in un linguaggio di programmazione (matlab, VBA o C++) è preferibile e riceverà una valutazione migliore.

4.9.6 Obiettivi dell'esercitazione

Di seguito sono elencate tutte le domande del tema d'esame. Ogni gruppo è chiamato a rispondere solo ad un sottoinsieme di queste domande, secondo le indicazioni riportate nel capitolo successivo.

4.9.6.a Domande sui modelli stocastici

(MO1) Nel caso del modello a salti descritto nell'equazione 4.56, quale condizione va posta sul parametro μ in modo tale che venga garantita la condizione di neutralità al rischio per dS/S.
Cosa accade se $\lambda = 0$?
Cosa accade nel modello se si considera $J_1 = J_2 = 1$?

(MO2) Nel caso del modello a salti descritto nell'equazione 4.58, quale condizione va posta sul parametro μ in modo tale che venga garantita la condizione di neutralità al rischio per dS/S.
Cosa accade se $\lambda_a = \lambda_b = 0$? Cosa accade nel modello se si considera $J_a = J_b = 1$?

(MO3) Calibrazione del modello a salti.
Relativamente al modello a salti rappresentato dalla formula 4.54, si effettui una calibrazione dei parametri del modello (ovvero λ, σ e J) rispetto alle quotazioni di mercato riportate nel data set seguente:

(D) Data set la calibrazione del modello a salti

```
# Valore dell'azione all'istante iniziale
S_0 = 100

# Valore del tasso risk free
r = 5 %

# Data di maturit\'a dell'opzione (espressa in anni)
T = 0.25 y

# Quotazione di mercato delle call (strike price, prezzo)

Strike price            Prezzo call


86.06916838             15.99728375
88.60061451             13.7706829
91.13206064             11.63694665
93.66350677              9.632311128
96.1949529               7.794039224
```

98.72639903	6.154688772
101.2578452	4.736882976
103.7892913	3.549945186
106.3207374	2.589132263
108.8521835	1.837378068
111.3836297	1.268790147
113.9150758	0.852842767
116.4465219	0.558292208
118.9779681	0.356170889
121.5094142	0.221614842

(MO4) Call-put parity
Si considerino le stime Monte Carlo dei prezzi di una call e di una put riferite allo stesso strike price, utilizzando lo stesso numero di scenari (N) e ricorrendo alla stessa sequenza di numeri casuali (quindi utilizzando lo stesso seme per il pricing della call e della put). Si domanda se tali prezzi, ottenuti dal Monte Carlo, soddisfino o meno la call-put parity in maniera esatta (e non solo nel limite $N \to \infty$).
Si motivi la risposta dandone adeguata spiegazione.

(MO5) Si derivi l'analogo della formula di BS per un'opzione digitale di tipo call ed un processo log-normale risolto nello schema di Eulero in cui si consideri un unico passo di ampiezza $\Delta t = T$. In altri termini si derivi l'analogo della formula di BS per un'opzione digitale di tipo call (vedi definizione 4.43) nell'ipotesi che il valore a scadenza dell'azione $S(T)$ sia dato da:

$$\frac{S(T) - S_0}{S_0} = r\,T + \sigma\,\sqrt{T}\,w \tag{4.61}$$

4.9.6.b Domande sul metodo Monte Carlo

Si faccia riferimento alla combinazione di pay-off/modello/dati di mercato/tipo di linguaggio assegnato per rispondere alle seguenti domande.

(MC1) implementare l'algoritmo Monte Carlo standard utilizzando la discretizzazione di Eulero del processo stocastico.
Si faccia riferimento alla combinazione di pay-off/modello/dati di mercato/tipo di linguaggio assegnato (vedi data set riportato alla fine del tema d'esame).

(MC2) Nel caso dell'approssimazione di Eulero per generare i cammini, può accadere che il prezzo dell'azione simulato ad una data futura sia negativo?
Detta δt la lunghezza temporale del singolo step Monte Carlo, è più

probabile che tale problema insorga quando δt è piccolo oppure grande? Per quale motivo?
È possibile correggere/modificare lo schema di Eulero in modo da evitare l'insorgere di questo problema? In altri termini quale ricetta correttiva si può applicare in modo da evitare la generazione di valori futuri negativi di $S(t)$?

(MC3) implementare l'algoritmo Monte Carlo standard utilizzando la formula integrale esatta relativamente al processo stocastico considerato.
Si faccia riferimento alla combinazione di pay-off/modello/dati di mercato/tipo di linguaggio assegnato (vedi data set riportato alla fine del tema d'esame).

(MC4) Calcolare l'andamento dell'errore Monte Carlo per diversi valori di N (ad es. in incrementi di 500/1000 scenari). Verificare che questo scali con N secondo la legge:

$$\epsilon = \frac{H(\sigma)}{\sqrt{N}} \tag{4.62}$$

dove $H(\sigma)$ rappresenta una semplice costante moltiplicativa (dipendente in generale dai dati di mercato, dal tipo di pay-off, ecc.). Nel caso dell'equazione sopra riportata, è stata evidenziata la sola dipendeza di H da σ.
Suggerimento: si consideri un grafico che riporti l'andamento dell'errore al variare di N in coordinate log-log.
Si faccia riferimento alla combinazione di pay-off/modello/dati di mercato/tipo di linguaggio assegnato (vedi data set riportato alla fine del tema d'esame).

(MC5) Come indicato, l'andamento dell'errore Monte Carlo scala con N tramite la legge (4.62). L'andamento con $1/\sqrt{N}$ è tipico ed intrinseco del Monte Carlo, viceversa la costante al numeratore, $H(\sigma)$, dipende: dal problema (ovvero dal pay-off), dai dati di mercato (tra cui ad esempio la volatilità) e dalla metodologia con cui il Monte Carlo viene implementato (ad esempio nel caso si utilizzino metodi di riduzione delle varianza; si veda a tal proposito il punto successivo).
Determinare l'andamento numerico di $H(\sigma)$ al variare della volatilità (ad es. considerando valori crescenti di volatilità).
Si faccia riferimento alla combinazione di pay-off/modello/dati di mercato/tipo di linguaggio assegnato (vedi data set riportato alla fine del tema d'esame).

(MC6) Al fine di ridurre gli errori statistici si utilizzi la tecnica della variabile antitetica (vedi note del corso).
Rispetto all'andamento dell'errore definito dall'eq(4.62), qual'e' l'impatto della variabile antitetica nella riduzione dell'errore? Ovvero: (a)

cambia la legge di scala $1/\sqrt{N}$? (b) Cambia il valore di $H(\sigma)$ e come? In generale si ripeta l'analisi dei punti (MC4) e (MC5) e la si confronti con i risultati precedentemente ottenuti.
Si faccia riferimento alla combinazione di pay-off/modello/dati di mercato/tipo di linguaggio assegnato (vedi data set riportato alla fine del tema d'esame).

(MC7) Nella simulazione Monte Carlo si consideri al posto di una voltatilità σ costante, una volatilità $\sigma(t_i, t_{i+1})$ variabile.
$\sigma(t_i, t_{i+1})$ rappresenta la volatilità del sottostante tra la data di fixing t_i e t_{i+1} (volatilità forward).
Si faccia riferimento alla combinazione di pay-off/modello/dati di mercato/tipo di linguaggio assegnato (vedi data set riportato alla fine del tema d'esame).

(MC8) Relativamente al caso del pay-off di un'opzione con barriera down & out, si calcoli tramite il metodo Monte Carlo la probabilità di tocco della barriera. Si valuti anche l'errore Monte Carlo di tale stima.

(MC9) Relativamente al caso del pay-off di un'opzione digitale di tipo corridor, ovvero con doppia barriera, (vedi equazione 4.48), si calcoli tramite il metodo Monte Carlo la probabilità che il sottostante (S) rimanga all'interno del corridoio. Si valuti anche l'errore di tale stima.

(MC10) Relativamente al caso del pay-off di un'opzione call/put, si confronti la convergenza dei prezzi Monte Carlo al valore esatto di Black e Scholes. Si richiede quindi come passo propedeutico, di implementare la formula esatta di BS per le opzioni plain vanilla.
Sempre relativamente a questo contesto (opzioni call/put) qual'è la differenza tra effettuare un Monte Carlo con più step (ovvero con m date intermedie di ampiezza $\delta_t = T/m$) al posto di un unico salto fino a T (ovvero: $m = 1$, $\delta_t = T$)? (Si risponda sia considerando uno schema di Eulero relativo alla domanda MC1, sia considerando lo schema esatto riferito alla domanda MC3.) Quale tra le due soluzioni risulta migliore in termini di precisione e/o di performance e/o di tempi di esecuzione (i.e. cpu time).

4.9.6.c Pay-off

(PO1) Cosa si può dire sul posizionamento reciproco del prezzo di una plain vanilla rispetto alla corrispondente versione con barriera down & out? In altri termini il prezzo della plain vanilla è maggiore o minore della versione con barriera? E per quali motivi?

(PO2) Si consideri un'opzione con barriera digitale pari a B (vedi definizione 4.47) e un'opzione corridor con barriera inferiore pari a B e barriera superiore pari a $B_u > B$ (vedi definizione 4.48).
Quale tra le due opzioni ha prezzo maggiore?
Sia P_{barrier} il prezzo dell'ozione con barriera e P_{corridor} quello dell'opzione corridor, quale e' il limite a cui tende P_{corridor} per $B_u \to +\infty$?

4.9.7 Tema d'esame

4.9.7.a Gruppo 1

(P1) Gruppo 1

Si risponda alle seguenti domande: MC1, MC2, MC3, MC4, MC5, MC6, MC10
Domande facoltative: MO3, MO4

Relativamente alla sezione relativa al metodo Monte Carlo, si faccia riferimento al seguente contesto pay-off/modello/dati di mercato/tipo di linguaggio:

- Pay-off: call e put (vedi equazione 4.42)
- Modello: lognormale standard (vedi equazione 4.51)
- Set di dati: set numero 1 (vedi sotto)
- Implementazione: almeno uno tra Excel, VBA o matlab. Riceverà una migliore valutazione un'implementazione basata su matlab o VBA.

I risulti ottenuti devono essere presentati sotto forma di breve discussione scritta, con l'ausilio eventuale di tabelle e grafici (eventualmente contenuti in un foglio excel se questo fosse ritenuto più comodo).

Si consideri il seguente set di dati (rispettivamente dati di mercato, dati anagrafici e dati relativi alla simulazione Monte Carlo). Il set riportato di seguito dovrà poi essere utilizzato durante l'esercitazione.

(D1) Data set 1

```
# Numero di simulazioni Monte Carlo
N = 100, 400, 1600 (> 10000 scenari solo nel caso si
    utilizzi VBA o matlab)

# Valore dell'azione all'istante iniziale
S_0 = 100

# Valore del tasso risk free
r = 5 %

# Valore della volatilita', su base annua, del sottostante
sigma = 10%, 15%, 25% e 30%

# Dati relativi all'opzione plain vanilla
#
E = 105

# Data di maturit\'a dell'opzione (espressa in anni)
T = 1y

# Numero di intervalli
num_interavalli = 1 o 12
```

4.9.7.b Gruppo 2

(P2) Gruppo 2

Si risponda alle seguenti domande: MC1, MC2, MC3, MC4, MC5, MC6
Domande facoltative: MO3, MO5

Relativamente alla sezione relativa al metodo Monte Carlo, si faccia riferimento al seguente contesto pay-off/modello/dati di mercato/tipo di linguaggio:

- Pay-off: digitale con barriera per 3 sottostanti (vedi equazione 4.46)
- Modello: lognormale multivariato per asset scorrelati (nello specifico $n = 3$) (vedi equazione 4.53)
- Set di dati: set numero 2 (vedi sotto)
- Implementazione: almeno uno tra Excel, VBA o matlab. Riceverà una migliore valutazione un'implementazione basata su matlab o VBA.

I risulti ottenuti devono essere presentati sotto forma di breve discussione scritta, con l'ausilio eventuale di tabelle e grafici (eventualmente contenuti in un foglio excel se questo fosse ritenuto più comodo).

Si consideri il seguente set di dati (rispettivamente dati di mercato, dati anagrafici e dati relativi alla simulazione Monte Carlo). Il set riportato di seguito dovrà poi essere utilizzato durante l'esercitazione.

(D2) Data set 2

```
# Numero di simulazioni Monte Carlo
N = 100, 400, 1600 (> 10000 scenari solo nel caso si
    utilizzi VBA o matlab)

# Valore dell'azione all'istante iniziale
S_1(t0) = 100
S_2(t0) = 50
S_3(t0) = 500

# Valore del tasso risk free
r = 5 %

# Valore della volatilita', su base annua, del sottostante
sigma_1 = 30%
sigma_2 = 10%
sigma_3 = 1%

# Dettagli dell'opzione con barriera su sottostanti multipli
#
K = 1
B = 70%, 80%, 90%, 99.9%

# Data di maturit\`a dell'opzione (espressa in anni)
T = 1y

# Numero di intervalli
num_interavalli = 12, 24
```

4.9.7.c Gruppo 3

(P3) Gruppo 3

Si risponda alle seguenti domande: MC1, MC2, MC3, MC4, MC5, MC6
Domande facoltative: MO4, MO5

Relativamente alla sezione relativa al metodo Monte Carlo, si faccia riferimento al seguente contesto pay-off/modello/dati di mercato/tipo di linguaggio:

- Pay-off: best of (vedi equazione 4.50)
- Modello: lognormale bivariato per due asset correlati (vedi equazione 4.52)
- Set di dati: set numero 3 (vedi sotto)
- Implementazione: almeno uno tra Excel, VBA o matlab. Riceverà una migliore valutazione un'implementazione basata su matlab o VBA.

I risulti ottenuti devono essere presentati sotto forma di breve discussione scritta, con l'ausilio eventuale di tabelle e grafici (eventualmente contenuti in un foglio excel se questo fosse ritenuto più comodo).

Si consideri il seguente set di dati (rispettivamente dati di mercato, dati anagrafici e dati relativi alla simulazione Monte Carlo). Il set riportato di seguito dovrà poi essere utilizzato durante l'esercitazione.

(D3) Data set 3

```
# Numero di simulazioni Monte Carlo
N = 100, 400, 1600 (> 10000 scenari solo nel caso si
    utilizzi VBA o matlab)

# Valore delle due azioni all'istante iniziale
S_1(t0) = 100
S_2(t0) = 50

# Valore del tasso risk free
r = 5 %

# Valore delle volatilita' delle due azioni, su base annua:
sigma_1 = 30%
sigma_2 = 2%

# Valore della correlazione
rho = -1, -0.5, 0, 0.5, 1

# Strike price dell'opzione (espresso in forma percentuale):
#
E = 80%, 90%, 110%, 120%

# Data di maturit\`a dell'opzione (espressa in anni)
T = 1y

# Numero di intervalli schema di Eulero
num_interavalli = 12, 24, 36

# Numero di intervalli schema esatto
num_interavalli = 1
```

4.9.7.d Gruppo 4

(P4) Gruppo 4

Si risponda alle seguenti domande: MC1, MC2, MC3, MC4, MC5, MC6, MC8, PO1, PO2
Domande facoltative: MO3, MO4, MO5

Relativamente alla sezione relativa al metodo Monte Carlo, si faccia riferimento al seguente contesto pay-off/modello/dati di mercato/tipo di linguaggio:

- Pay-off: digitale con barriera su due sottostanti ($n = 2$) (vedi equazione 4.46)
- Modello: lognormale bivariato per due asset correlati (vedi equazione 4.52)
- Set di dati: set numero 4 (vedi sotto)
- Implementazione: almeno uno tra Excel, VBA o matlab. Riceverà una migliore valutazione un'implementazione basata su matlab o VBA.

I risulti ottenuti devono essere presentati sotto forma di breve discussione scritta, con l'ausilio eventuale di tabelle e grafici (eventualmente contenuti in un foglio excel se questo fosse ritenuto più comodo).

Si consideri il seguente set di dati (rispettivamente dati di mercato, dati anagrafici e dati relativi alla simulazione Monte Carlo). Il set riportato di seguito dovrà poi essere utilizzato durante l'esercitazione.

(D4) Data set 4

```
# Numero di simulazioni Monte Carlo
N = 100, 400, 1600 (> 10000 scenari solo nel caso si
    utilizzi VBA o matlab)

# Valore delle due azioni all'istante iniziale
S_1(t0) = 200
S_2(t0) = 400

# Valore del tasso risk free
r = 5 %

# Valore delle volatilita' delle due azioni, su base annua:
sigma_1 = 30%, 20%
sigma_2 = 1%,  40%

# Valore della correlazione
rho = -1, -0.5, 0, 0.5, 1

# Dati relativi all'opzione digitale con barriera su due sottostanti
#
K = 1
B = 10%, 50%, 90%, 99%

# Data di maturit\`a dell'opzione (espressa in anni)
T = 1y

# Numero di intervalli
num_interavalli = 6, 12, 24, 36, 48
```

4.9.7.e Gruppo 5

(P5) Gruppo 5

Si risponda alle seguenti domande: MO1, MC1, MC2, MC4, MC5, MC6, MC9, PO2
Domande facoltative: MO4, MO5

Relativamente alla sezione relativa al metodo Monte Carlo, si faccia riferimento al seguente contesto pay-off/modello/dati di mercato/tipo di linguaggio:

- Pay-off: corridor digitale (vedi equazione 4.48)
- Modello: a salti con ampiezza variabile (vedi equazione 4.56)
- Set di dati: set numero 5 (vedi sotto)
- Implementazione: almeno uno tra Excel, VBA o matlab. Riceverà una migliore valutazione un'implementazione basata su matlab o VBA.

I risulti ottenuti devono essere presentati sotto forma di breve discussione scritta, con l'ausilio eventuale di tabelle e grafici (eventualmente contenuti in un foglio excel se questo fosse ritenuto più comodo).

Si consideri il seguente set di dati (rispettivamente dati di mercato, dati anagrafici e dati relativi alla simulazione Monte Carlo). Il set riportato di seguito dovrà poi essere utilizzato durante l'esercitazione.

(D5) Data set 5

```
# Numero di simulazioni Monte Carlo
N = 100, 400, 1600 (> 10000 scenari solo nel caso si
    utilizzi VBA o matlab)

# Valore dell'azione all'istante iniziale
S_0 = 100

# Valore del tasso risk free
r = 5 %

# Valore della volatilita', su base annua, del sottostante
sigma = 10%, 15%, 25%, 30% e 50%

# Livelli delle due barriere per l'opzione corridor e valore di K
#
B_l = 90
B_u = 110
K   = 1

# Per il processo a salto ad ampiezza variabile
#
lambda = 0.4
J_1 = 0.5
J_2 = 1.5
p = 90%

# Data di maturit\`a dell'opzione (espressa in anni)
T = 1y

# Numero di intervalli
num_interavalli = 12
```

4.9.7.f Gruppo 6

(P6) Gruppo 6

Si risponda alle seguenti domande: MO2, MC1, MC2, MC4, MC5, MC6
Domande facoltative: MO3, MO4, MC3

Relativamente alla sezione relativa al metodo Monte Carlo, si faccia riferimento al seguente contesto pay-off/modello/dati di mercato/tipo di linguaggio:

- Pay-off: reverse cliquet (vedi equazione 4.49)
- Modello: a doppio salto (vedi equazione 4.58)
- Set di dati: set numero 6 (vedi sotto)
- Implementazione: almeno uno tra Excel, VBA o matlab. Riceverà una migliore valutazione un'implementazione basata su matlab o VBA.

I risulti ottenuti devono essere presentati sotto forma di breve discussione scritta, con l'ausilio eventuale di tabelle e grafici (eventualmente contenuti in un foglio excel se questo fosse ritenuto più comodo).

Si consideri il seguente set di dati (rispettivamente dati di mercato, dati anagrafici e dati relativi alla simulazione Monte Carlo). Il set riportato di seguito dovrà poi essere utilizzato durante l'esercitazione.

(D6) Data set 6

```
# Numero di simulazioni Monte Carlo
N = 100, 400, 1600 (> 10000 scenari solo nel caso si
    utilizzi VBA o matlab)

# Valore dell'azione all'istante iniziale
S_0 = 200

# Valore del tasso risk free
r = 4 %

# Valore della volatilita', su base annua, del sottostante
sigma = 15%, 25%, 30% e 40%

# Dati per l'opzione reverse cliquet
#
L = 4%
H = 12%

# Per il processo a doppio salto
#
lambda_a = 0.5
lambda_b = 0.1
J_a = 0.5
J_b = 1.5

# Data di maturit\`a dell'opzione (espressa in anni)
T = 1y

# Numero di intervalli
num_interavalli = 12
```

4.9.7.g Gruppo 7

(P7) Gruppo 7

Si risponda alle seguenti domande: MC1, MC2, MC3, MC4, MC5, MC6, MC7, PO2
Domande facoltative: MO3, MO5

Relativamente alla sezione relativa al metodo Monte Carlo, si faccia riferimento al seguente contesto pay-off/modello/dati di mercato/tipo di linguaggio:

- Pay-off: digitale con barriera (vedi equazione 4.47)
- Modello: lognormale standard (vedi equazione 4.51)
- Set di dati: set numero 7 (vedi sotto)
- Implementazione: almeno uno tra Excel, VBA o matlab. Riceverà una migliore valutazione un'implementazione basata su matlab o VBA.

I risulti ottenuti devono essere presentati sotto forma di breve discussione scritta, con l'ausilio eventuale di tabelle e grafici (eventualmente contenuti in un foglio excel se questo fosse ritenuto più comodo).

Si consideri il seguente set di dati (rispettivamente dati di mercato, dati anagrafici e dati relativi alla simulazione Monte Carlo). Il set riportato di seguito dovrà poi essere utilizzato durante l'esercitazione.

(D7) Data set 7

```
# Numero di simulazioni Monte Carlo
N = 100, 400, 1600 (> 10000 scenari solo nel caso si
    utilizzi VBA o matlab)

# Valore dell'azione all'istante iniziale
S_0 = 100

# Valore del tasso risk free
r = 5 %

# Valore della volatilita', su base annua, del sottostante
sigma = 10%, 15%, 25% e 30%

# Dati relativi all'opzione digitale con barriera
#
K = 1
B = 20, 90, 95, 99 e 120

# Data di maturit\`a dell'opzione (espressa in anni)
T = 1y

# Numero di intervalli
num_interavalli = 12
```

4.9.7.h Gruppo 8

(P8) Gruppo 8

Si risponda alle seguenti domande: MC1, MC2, MC3, MC4, MC5, MC6, MC7, PO2
Domande facoltative: MO3, MO4, MO5

Relativamente alla sezione relativa al metodo Monte Carlo, si faccia riferimento al seguente contesto pay-off/modello/dati di mercato/tipo di linguaggio:

- Pay-off: digitale con barriera (vedi equazione 4.47)
- Modello: lognormale standard (vedi equazione 4.51)
- Set di dati: set numero 8 (vedi sotto)
- Implementazione: almeno uno tra Excel, VBA o matlab. Riceverà una migliore valutazione un'implementazione basata su matlab o VBA.

I risulti ottenuti devono essere presentati sotto forma di breve discussione scritta, con l'ausilio eventuale di tabelle e grafici (eventualmente contenuti in un foglio excel se questo fosse ritenuto più comodo).

Si consideri il seguente set di dati (rispettivamente dati di mercato, dati anagrafici e dati relativi alla simulazione Monte Carlo). Il set riportato di seguito dovrà poi essere utilizzato durante l'esercitazione.

(D8) Data set 8

```
# Numero di simulazioni Monte Carlo
N = 100, 400, 1600 (> 10000 scenari solo nel caso si
    utilizzi VBA o matlab)

# Valore dell'azione all'istante iniziale
S_0 = 20

# Valore del tasso risk free
r = 1 %

# Valore della volatilita', su base annua, del sottostante
sigma = 20%, 25%, 30% e 50%

# Dati relativi all'opzione digital barrier
#
K = 1
B = 2, 10, 18, 19 e 30

# Data di maturit\`a dell'opzione (espressa in anni)
T = 1y

# Numero di intervalli
num_interavalli = 12
```

(D8) Integrazione al data set 8 da utilizzare x il punto MC7

```
# Valore della volatilita' forward x il sottostante

intervallo temporale                    Volatilita' (forward)


01 intervallo                            35%
02 intervallo                            32%
03 intervallo                            30%
04 intervallo                            28%
05 intervallo                            27%
06 intervallo                            26%
07 intervallo                            24%
08 intervallo                            20%
09 intervallo                            18%
10 intervallo                            18%
11 intervallo                            18%
12 intervallo                            18%
```

4.9.7.i Gruppo 9

(P9) Gruppo 9

Si risponda alle seguenti domande: MC1, MC2, MC3, MC4, MC5, MC6, PO1
Domande facoltative: PO2, MO4

Relativamente alla sezione relativa al metodo Monte Carlo, si faccia riferimento al seguente contesto pay-off/modello/dati di mercato/tipo di linguaggio:

- Pay-off: digital call (vedi equazione 4.43)
- Modello: lognormale standard (vedi equazione 4.51)
- Set di dati: set numero 9 (vedi sotto)
- Implementazione: almeno uno tra Excel, VBA o matlab. Riceverà una migliore valutazione un'implementazione basata su matlab o VBA.

I risulti ottenuti devono essere presentati sotto forma di breve discussione scritta, con l'ausilio eventuale di tabelle e grafici (eventualmente contenuti in un foglio excel se questo fosse ritenuto più comodo).

Si consideri il seguente set di dati (rispettivamente dati di mercato, dati anagrafici e dati relativi alla simulazione Monte Carlo). Il set riportato di seguito dovrà poi essere utilizzato durante l'esercitazione.

(D9) Data set 9

```
# Numero di simulazioni Monte Carlo
N = 100, 400, 1600 (> 10000 scenari solo nel caso si
    utilizzi VBA o matlab)

# Valore dell'azione all'istante iniziale
S_0 = 500

# Valore del tasso risk free
r = 2 %

# Valore della volatilita', su base annua, del sottostante
sigma = 15%, 25%, 30% e 50%

# Dati relativi all'opzione digital call
#
E = 500

# Data di maturit\`a dell'opzione (espressa in anni)
T = 1y

# Numero di intervalli
num_interavalli = 1 o 12
```

Capitolo 5

Elementi di programmazione ad oggetti

5.1 Introduzione alla programmazione ad oggetti

Il presente capitolo intende fornire una guida sintetica all'apprendimento dei concetti base della programmazione ad oggetti in C++ con particolare riferimento ai problemi di carattere numerico.

Conoscenze necessarie

Si presuppone, nei partecipanti, una conoscenza matematica adeguata ai temi trattati.

Struttura delle dispense

La presente sezione si compone di tre capitoli fondamentali:

- Una parte introduttiva dedicata ai fondamenti del C++ (sezione 5.2), il cui scopo è quello di definire i concetti base del linguaggio: tipi variabili, costrutti fondamentali (if, ciclo for ecc.), utilizzo di funzioni ecc.
- Una parte dedicata alla programmazione ad oggetti in senso compiuto. (sezione 5.3), in cui verranno presentate le idee alla base della programmazione ad oggetti, ovvero:
  a) incapsulamento e mascheramento;
  b) polimorfismo
  c) ereditarietà.
- Breve introduzione ai "design patterns".

Bibliografia

Alla fine delle note è riportata una breve bibliografia di libri di testo sull'argomento. In particolare si suggeriscono alcuni testi di riferimento: [25, 26], alcuni testi su argomenti avanzati: [27] e del materiale disponibile su web: [28, 29, 30].

5.2 Il contesto - Elementi base del C++

In questo primo capitolo definiremo alcuni elementi base del linguaggio C++. In particolare:

- Introduzione alla programmazione procedurale e ad oggetti: differenze e vantaggi.
- Elementi base del C/C++: tipi variabili fondamentali, condizioni, cicli for, gestione input/output, funzioni ecc.
- Compilatori e ambienti di compilazione per la gestione di progetti.
- Strutturazione di una libreria: nomenclatura dei filename e organizzazione dei file.
- Esempi di scrittura di semplici programmi in C++.

5.2.1 Programmazione procedurale e ad oggetti

- Programmazione procedurale: un problema complesso è scomposto in una serie di task più semplici, per ciascuno di questi viene scritta una funzione in grado di risolverlo. Assemblando le varie funzioni, dedicate alla risuluzione dei task elementari, si ottiene la risoluzione del task complesso.
  Caratteristica di questo stile di programmazione è il focus sulle procedure/funzioni.
  Questo modo di procedere permette una certa riutilizzabilità del software (anche se non paragonabile a quella permessa dalla programmazione ad oggetti). Infatti le singole funzioni, se ben progettate, possono essere utilizzate anche in altri contesti. Si pensi ad esempio ad una funzione per integrare numericamente una funzione reale di variabile reale, su un intervallo definito. Tale funzione può trovare applicazione in una miriade di contesti (ovvero di problemi) differenti.
  Gli svantaggi della programmazione procedurale sono fondamentalmente i seguenti:

  (a) l'utlizzatore di una funzione che risolve un task elementare, deve sapere quali parametri inviare alla funzione e quale e' il loro significato. In altri termini deve avere una conoscenza specifica del funzionamento della funzione.

  (b) Nella programmazione procedurale le funzioni ed i dati (che vengono passati alle funzioni) sono logicamente e fisicamente distinti, questo produce rischi ed errori (se ad es. viene passato un dato nel formato sbagliato ad una funzione). Ricordiamo che nella programmazione procedurale il focus è sulle funzioni e non sui dati.

(c) La programmazione procedurale non è "naturale", nel senso che gli esseri umani tendono a strutturare il proprio ragionamento logico per classi di oggetti tra loro interagenti (vedi punto successivo) e non per strutture di task.

- Programmazione ad oggetti: il problema viene rappresentato tramite una visione ad oggetti. Ciascun elemento del mondo viene rappresentato da un oggetto. La soluzione del problema si ottiene in modo naturale, facendo interagire opportunamente gli oggetti del mondo. Cio' che caratterizza la programmazione ad oggetti è:

  – l'incapsulamento di dati e funzioni in un'unica classe
  – il polimorfismo ("overloading")
  – l'ereditarieta'

  I vantaggi offerti dalla programmazione ad oggetti sono speculari rispetto ai punti deboli della programmazione procedurale, ovvero:

  (a) nella programmazione ad oggetti, l'utilizzatore di un oggetto non è tenuto a sapere come l'oggetto è stato implementato al proprio interno. L'utente interagisce con l'oggetto unicamente attraverso la sua interfaccia.

  (b) nella programmazione ad oggetti dati e funzioni vengono incapsulate in un unico costrutto, appunto l'oggetto. Questo impedisce di avere errori o funzionamenti imprevisti causati dal passare un dato "errato" ad una funzione.

  (c) la programmazione ad oggetti è naturale, nel senso che riproduce ed è basata sullo stesso modo di strutturare il ragionamento degli esseri umani.

  (d) come conseguenza dei punti elencati sopra, la programmazione ad oggetti produce del codice estremamente modulare e facilmente riutilizzabile. Questo è un punto chiave nella creazione del software. Infatti la riutilizzabilità del codice permette di: i) ridurre i tempi di sviluppo ed i costi e ii) produrre del software più sicuro (gli oggetti possono essere manipolati solo attraverso la loro interfaccia, questo riduce drasticamente la possibilità di uso non consono del codice).

  Questa tecnica di programmazione è stata sviluppata negli anni 80. I principali linguaggi ad oggetti sono C++ e Java.

Nei paragrafi successivi verranno definiti i concetti fondamentali della programmazione in C (quindi ancora una programmazione di tipo procedurale).

5.2.2 Perché il C++

I vantaggio offerti dal C++ sono i seguenti:

- Supporta la programmazione ad oggetti.
- È compatibile (quasi al 100%) con il C.
- Permette un controllo anche a basso livello della macchina (proprietà derivata dal C).
- Non è un linguaggio interpretato (ovvero consente la creazione di un codice binario per l'esecuzione del programma).

Va osservato che il C++ presenta alcuni svantaggi:

- Non vengono fatte verifiche dal linguaggio sull'eventuale sforamento dei vettori.
- Non possiede alcune caratteristiche avanzate di alcuni linguaggi ad oggetti (come ad esempio la capacità di introspezione da parte degli oggetti).

Il Java permette ad esempio di superare questi aspetti negativi. In questo senso è un linguaggio più potente, sicuro e moderno del C++. Consente anche una maggiore integrazione con il web ed è più facilmente portabile su piattaforme differenti. Paga tutto questo con una minore velocità in fase di esecuzione ed essendo fortemente astratto rispetto alla macchina fisica, non consente alcune operazioni di basso livello che il C++ invece permette.

5.2.3 Tipi variabili fondamentali e concetti base

Terminatore di fine istruzione: ; Commenti: //

- int - numero intero
- float - numero reale in singola precisione
- double - numero reale in doppia precisione
- char - carattere alfanumerico

In C++ ogni variabile è caratterizzata dalle seguenti proprietà:

- nome: una stringa (identificatore) che referenzia la variabile in oggetto
- tipo: la tipologia a cui la variabile appartiene, ad es. int, float, double ecc.

- scopo: l'area di codice all'interno della quale la variabile esiste (scopo di file, scopo di classe ecc.)
- indirizzo: l'indirizzo di memoria dove la variabile è allocata
- dimensioni: l'ammontare di memoria (in byte) necessaria per immagazzinare la variabile in oggetto. Si noti che l'effettiva precisione (ovvero il numero di digit significativi, ovvero la dimensione della memoria allocata) di un float o di un double, dipende dalla particolare macchina, ovvero non è fissato a priori dal linguaggio. Ad es. valori tipici per le variabili di tipo int sono 4 byte, mentre per i double sono 8 byte.

5.2.3.a Programma d'esempio sull'utilizzo di variabili

- Nome file: main.cpp
- Contenuto: esempio di utilizzo di variabili fondamentali

main.cpp

```
// ------------------- Begin esempio_1 ---------------
//
// Tipi variabili fondamentali


int main(void) {

   int n ;      // n dichiarato

   n = 2 ;      // n valorizzato

   int m = 2 ;  // m dichiarato ed inizializzato

   double d = 10.5 ;

   char c = 'a' ;

}


// ------------------ End esempio_1 -----------------
```

Commenti

(1) si noti il carattere di fine istruzione: ;

5.2.4 Input output

Le funzioni per la gestione dell'input/output sono in C++: cin e cout.
Importante: includere sempre all'inzio del file l'istruzione di include:
#include $< iostream >$

5.2.4.a Programma d'esempio sull'utilizzo di cin e cout

- Nome file: main.cpp
- Contenuto: esempio di utilizzo delle funzioni cin e cout

main.cpp

```
// ------------------ Begin esempio_2 ---------------
//
// Input e output


#include <iostream>

using namespace std;

int main(void) {

   int n ;

   char str[80] ;

   // Stampa la frase come un printf()
   //
   cout << "Introdurre il numero di cui calcolare il fattoriale: " ;

   // Legge il valore e lo copia in n come scanf()
   //
   cin >> n;

   cout << "Introdurre una stringa: ";
   cin >> str;                      // Legge la stringa copiandola in str

   cout << "Il numero introdotto e': " << n << " mentre str = " << str << "\n";

}


// ------------------ End esempio_2 -----------------
```

Commenti

(1) Con le funzioni di *cin* e *cout* si possono trattare allo stesso modo stringhe, int, double, senza mai preoccuparsi di specificare il tipo e il numero di caratteri da stampare (a differenza di quanto accade con

scanf e *printf*). L'operatore << in questo contesto rappresenta un operatore di output. Mentre >> indica in questo caso l'input da tastiera. Da notare che in C++ e' sempre possibile usare *scanf()* e *printf()*, le funzioni per la gestione dell'input/output native del C.

5.2.5 Ciclo for

Il costrutto per effettuare un ciclo è: *for*. Questo costrutto permette di eseguire un certo numero di volte un blocco di istruzioni.

5.2.5.a Programma d'esempio sull'utilizzo del for

- Nome file: main.cpp
- Contenuto: esempio di utilizzo di un ciclo for

main.cpp

```
// ------------------ Begin esempio_3 ---------------
//
// Ciclo for



#include <iostream>

using namespace std;

int main(void) {


   // Lettura dati di input
   //
   int n ;
   char str[80] ;

   // Stampa la frase come un printf()
   //
   cout << "Introdurre il numero di cui calcolare il fattoriale: ";

   // Legge il valore e lo copia in n come scanf()
   //
   cin >> n;


   cout << "\n";
   cout << "Introdurre una stringa: ";
   cin >> str;            // Legge la stringa copiandola in str

   cout << "\n";
   cout << "Il numero introdotto e': " << n << " mentre str = " << str << "\n";


   // Calcolo del fattoriale
   //
   int fattoriale ;

   fattoriale = 1 ;
```

```
   for(int i=0; i<n; i++) {
      fattoriale = fattoriale * (i+1) ;

   }


   // Visualizzazione risultato
   //
   cout << "Il fattoriale di " << n << " e': " << fattoriale << "\n";

}



// ------------------ End esempio_3 -----------------
```

5.2.6 Costrutto while

Questo costrutto permette di ripetere un set di istruzioni fino a quando una condizione non risulta verificata. La parola chiaver per utilizzare questo costrutto è: *while*.

5.2.6.a Programma d'esempio sull'utilizzo del costrutto while

- Nome file: main.cpp
- Contenuto: esempio di utilizzo del costrutto while

main.cpp

```
// ------------------ Begin esempio_4 ---------------
//
// Costrutto while



#include <iostream>

using namespace std;

int main(void) {


   // Lettura dati di input
   //
   int n ;
   char str[80] ;

   // Stampa la frase come un printf()
   //
   cout << "Introdurre il numero di cui calcolare il fattoriale: ";

   // Legge il valore e lo copia in n come scanf()
   //
   cin >> n;


   cout << "\n";
   cout << "Introdurre una stringa: ";
   cin >> str;            // Legge la stringa copiandola in str

   cout << "\n";
   cout << "Il numero introdotto e': " << n << " mentre str = " << str << "\n";


   // Calcolo del fattoriale
   //
   int i = 1 ;
   int fattoriale = 1 ;
```

```
   while(i<=n) {
      fattoriale = fattoriale * i ;
      i ++ ;
   }


   // Visualizzazione risultato
   //
   cout << "Il fattoriale di " << n << " e': " << fattoriale << "\n";

}



// ------------------ End esempio_4 -----------------
```

5.2.7 Condizione if

Il costrutto per eseguire un set di istruzioni se una certa condizione risulta verificata è: *if*.
La condizione *if* può essere seguita da *else*. In tal modo è possibile eseguire un certo blocco di istruzioni se (*if*) è verificata una certa condizioni, diversamente (*else*) viene eseguito un altro blocco di istruzioni.

5.2.7.a Programma d'esempio sull'uso di condizioni

- Nome file: main.cpp
- Contenuto: esempio di utilizzo del costrutto if

main.cpp

```
// ------------------ Begin esempio_5 ---------------
//
// Utilizzo di if


#include <cstdlib>
#include <iostream>

using namespace std;

int main(void) {


   // Lettura dati di input
   //
   int n ;
   char str[80] ;

   // Stampa la frase come un printf()
   //
   cout << "Introdurre il numero di cui calcolare il fattoriale: ";

   // Legge il valore e lo copia in n come scanf()
   //
   cin >> n;


   cout << "\n";
   cout << "Introdurre una stringa: ";
   cin >> str;            // Legge la stringa copiandola in str

   cout << "\n";
   cout << "Il numero introdotto e': " << n << " mentre str = " << str << "\n";



   // Controllo dati di input
   //
   if(n<0) {
      // Errore nell'input
```

```
      //
      cout << "input n deve essere >=0. n = " << n << "\n" ;
      exit(1) ;
   }



   // Calcolo del fattoriale
   //
   int i = 1 ;
   int fattoriale = 1 ;

   while(i<=n) {
      fattoriale = fattoriale * i ;
      i ++ ;
   }


   // Visualizzazione risultato
   //
   cout << "Il fattoriale di " << n << " e': " << fattoriale << "\n";

}



// ------------------ End esempio_5 -----------------
```

5.2.8 Definizione ed utilizzo di una funzione

Una funzione è un insieme di istruzioni elementari, il cui scopo è risolvere un certo task. Una funzione càratterizzata da un nome, dall'insieme (tipologia) dei dati di input che necessita e dall'output che restituisce.

Il codice riportato sotto mostra l'utilizzo di una funzione.

5.2.8.a Programma d'esempio sull'uso di funzioni

- Nome file: main.cpp
- Contenuto: esempio di utilizzo di una funzione

main.cpp

```
// ------------------ Begin esempio_6 ---------------
//
// Utilizzo di una funzione


#include <cstdlib>
#include <iostream>

using namespace std;

int Calcolo_fattoriale(int n) ;

int main(void) {


   // Lettura dati di input
   //
   int n ;
   char str[80] ;

   // Stampa la frase come un printf()
   //
   cout << "Introdurre il numero di cui calcolare il fattoriale: ";

   // Legge il valore e lo copia in n come scanf()
   //
   cin >> n;


   cout << "\n";
   cout << "Introdurre una stringa: ";
   cin >> str;            // Legge la stringa copiandola in str

   cout << "\n";
   cout << "Il numero introdotto e': " << n << " mentre str = " << str << "\n";


   // Controllo dati di input
   //
   if(n<0) {
```

```
      // Errore nell'input
      //
      cout << "input n deve essere >=0. n = " << n << "\n" ;
      exit(1) ;
   }


   // Visualizzazione risultato
   //
   cout << "Il fattoriale di " << n << " e': " << Calcolo_fattoriale(n) << "\n";

}


int Calcolo_fattoriale(int n) {

   // Calcolo del fattoriale
   //
   int fattoriale ;

   fattoriale = 1 ;

   for(int i=0; i<n; i++) {
      fattoriale = fattoriale * (i+1) ;

   }

   return fattoriale ;

}

// ------------------ End esempio_6 -----------------
```

Meglio in questi casi separare la funzione in un file distinto da quello che contiene il main.

5.2.8.b Programma d'esempio sull'uso di funzioni II

- Nome file: main.cpp, calcolo_fattoriale.cpp e calcolo_fattoriale.hpp
- Contenuto: esempio di utilizzo di una funzione

main.cpp

```
// ------------------ Begin esempio_7 ---------------
//
// Utilizzo di una funzione


#include <cstdlib>
#include <iostream>

#include "calcolo_fattoriale.hpp"


using namespace std;

int main(void) {


   // Lettura dati di input
   //
   int n ;
   char str[80] ;

   // Stampa la frase come un printf()
   //
   cout << "Introdurre il numero di cui calcolare il fattoriale: ";

   // Legge il valore e lo copia in n come scanf()
   //
   cin >> n;


   cout << "\n";
   cout << "Introdurre una stringa: ";
   cin >> str;            // Legge la stringa copiandola in str

   cout << "\n";
   cout << "Il numero introdotto e': " << n << " mentre str = " << str << "\n";


   // Controllo dati di input
   //
```

```
   if(n<0) {
      // Errore nell'input
      //
      cout << "input n deve essere >=0. n = " << n << "\n" ;
      exit(1) ;
   }


   // Visualizzazione risultato
   //
   cout << "Il fattoriale di " << n << " e': " << Calcolo_fattoriale(n) << "\n";

}


// ------------------ End esempio_7 -----------------
```

calcolo_fattoriale.cpp

```
// ------------------ Begin esempio_7 ---------------
//
// Utilizzo di una funzione




int Calcolo_fattoriale(int n) {

   // Calcolo del fattoriale
   //
   int fattoriale ;

   fattoriale = 1 ;

   for(int i=0; i<n; i++) {
      fattoriale = fattoriale * (i+1) ;

   }

   return fattoriale ;

}

// ------------------ End esempio_7 -----------------
```

calcolo_fattoriale.hpp

```
// ------------------ Begin esempio_7 ---------------
```

```
//
// Utilizzo di una funzione


#ifndef _CALCOLO_FATTORIALE_HPP
#define _CALCOLO_FATTORIALE_HPP

int Calcolo_fattoriale(int n) ;


#endif


// ------------------- End esempio_7 -----------------
```

5.2.9 Tipi derivati

I tipi derivati sono dei tipi ottenuti a partire dai tipi dati predefiniti nel linguaggio, attraverso:

(a) l'uso degli operatori: [] (vettori o array), * (pointer) e & (reference)

(b) definendo delle strutture (struct)

Sono tipi derivati ad esempio:

- i vettori o le matrici (ad es. double vect[10] o double matrix[10][10]);
- le stringhe (ad es. char name[10]);
- i puntatori a variabili (ad es. double *pointer) ;
- le strutture.

I tipi derivati sono quindi ottenuti dai tipi primitivi tramite composizione. Un tipo primitivo non può essere decomposto in tipi più elementari. I tipi primitivi fondamentali sono stati illustrati nei preedenti paragrafi (ad. es. double, float, int, char ecc. ecc.).

5.2.10 Tipi derivati: i puntatori

Un tipo derivato in C++ molto interessante (e non presente in altri linguaggi come il fortran o il visual basic) è dato dai puntatori.

5.2.10.a Programma d'esempio sull'utilizzo dei puntatori

- Nome file: main.cpp
- Contenuto: esempio di utilizzo di un puntatore

main.cpp

```
// ------------------ Begin esempio_8 ---------------
//
// Utilizzo di un puntatore


#include <iostream>

using namespace std;

int main(void) {

   // Lettura dati di input
   //
   double *pointer ;

   double a ;

   a = 10 ;
   pointer = &a ;


   cout << "contenuto area di memoria indirizzata da pointer = " << *pointer << "\n" ;

   *pointer = 20 ;


   // Visualizzazione risultato
   //
   cout << "contenuto area di memoria indirizzata da pointer = " << *pointer << "\n" ;
   cout << "contenuto di a = " << a << "\n" ;

}

// ------------------ End esempio_8 -----------------
```

5.2.10.b Programma d'esempio sull'uso dei puntatori nelle funzioni

- Nome file: main.cpp
- Contenuto: esempio di utilizzo di un puntatore in una funzione

main.cpp

```
// ------------------ Begin esempio_9 ---------------
//
// Utilizzo di un puntatore in una funzione


#include <iostream>

using namespace std;


int Calcola_perimetro_e_area_quadrato(double lato, double *perimetro, double *area) ;

int main(void) {


   double lato ;

   // Lettura dati di input
   //
   cout << "Introdurre il lato del quadrato: "; // Stampa la frase come un printf()
   cin >> lato;


   double perimetro = 0 ;
   double area = 0 ;
   int stato ;

   stato = Calcola_perimetro_e_area_quadrato(lato, &perimetro, &area) ;

   if(stato == 0) {
      // Visualizzazione risultato
      //
      cout << "perimetro = " << perimetro << " area = " << area << "\n" ;
   }
   else {
      cout << "errore nell'elaborazione \n" ;
   }
```

```
}


int Calcola_perimetro_e_area_quadrato(double lato, double *perimetro, double *area) {

   if(lato >=0 ) {
      *perimetro = 4 * lato ;
      *area = lato * lato ;
      return 0 ;
   }
   else {
      *perimetro = 0 ;
      *area = 0 ;
      return 1 ;
   }


}

// ------------------ End esempio_9 -----------------
```

Commenti

(1) Anche in questo caso è più pulito implementare la funzione in un file separato.

5.2.11 Tipi derivati: i vettori

Un altro tipo derivato è costituito dai vettori (array). Un array è un insieme di dati dello stesso tipo, referenziato tramite un unico identificatore.
L'esempio riportato di seguito chiarisce la definizione e l'utilizzo degli array. Nell'esempio sono riportate due modalità differenti nella definizione dei vettori:

(1) Allocazione della memoria da parte del sistema operativo.
Ad es. double vector[10] ;
In questo caso la gestione della memoria viene demandata e gestita in automatico dal programma / sistema operativo. Se ad esempio la definizione del vettore avviene all'interno di una funzione, la memoria verrà allocata al momento della definizione nello stack segment (vedi paragrafo sulla gestione della memoria in C). La memoria verrà poi deallocata in automatico quando la variabile uscirà di scopo (ad es. quando si esce dalla funzione).
L'utente non dovrà quindi preoccuparsi di gestire l'allocazione / deallocazione della memoria.
La dimensione del vettore deve però essere nota in fase di compilazione.

(2) Allocazione dinamica della memoria tramite l'utilizzo di un puntatore.
Ad es. double * pointer ; pointer = new double[n] ;
In questo caso la gestione della memoria viene demandata interamente all'utente che è responsabile dell'allocazione della memoria tramite l'operatore *new* e la successiva deallocazione tramite *delete*. La memoria necessaria ad immagazzinare il vettore viene rieservata all'interno dello heap (vedi paragrafo sulla gestione della memoria in C). Si noti invece come il pointer che contiene l'indirizzo del vettore, essendo tipicamente una variabile definita all'interno di una funzione, trova la sua allocazione all'interno dello stak segment ed uscirà di scopo (quindi verrà deallocata automaticamente dal sistema operativo) al termine della funzione.
Se l'utente non dealloca la memoria prima che la variabile puntatore esca di scopo, tale memoria rimarrà residente nella RAM senza più poter essere deallocata (si ha in questo caso un cosiddetto memory leak).

5.2.11.a Programma d'esempio sull'utilizzo dei vettori

- Nome file: main.cpp
- Contenuto: esempio di utilizzo di un vettore

main.cpp

```
// ------------------ Begin esempio_10 ---------------
//
// Utilizzo di un vettore
//

#include <iostream>

using namespace std;

int main(void) {


   // Lettura dati di input
   //
   int n ;

   cout << "Introdurre la dimensione del vettore: "; // Stampa la frase come un printf()
   cin >> n;


   double vector[n] ;

   for(int i=0; i<n; i++) {
      vector[i] = i ;
   }


   // Allocazione dinamica della memoria
   //
   double *pointer_vector ;

   pointer_vector = new double[n] ;

   for(int i=0; i<n; i++) {
      pointer_vector[i] = 100+ i ;
   }


   for(int i=0; i<n; i++) {
      cout << "contenuto di vector[i] = " << vector[i] "\n"  ;
```

```
      cout << "contenuto di pointer_vector[i] = " << pointer_vector[i] << "\n"  ;
   }


   delete [] pointer_vector ;

}

// ------------------ End esempio_10 -----------------
```

Si noti che la dimensione dell'array deve essere un'espressione costante (quindi fissata e nota già in fase di compilazione). Non può quindi essere una grandezza variabile definita in fase di run-time. L'allocazione di memoria viene eseguita dal compilatore prima dell'esecuzione del programma.

Si consideri ora il seguente problema: si definiscano due vettori di uguale lunghezza e si costruisca una funzione che li scambi (ovvero che scambi gli elementi dell'uno con quelli dell'altro).

5.2.11.b Programma d'esempio sullo swap di due vettori

- Nome file: main.cpp
- Contenuto: swap tra due vettori

main.cpp

```
// ------------------ Begin esempio_11 ---------------
//
// Swap tra due vettori
//



#include <iostream>

using namespace std;

int main(void) {


   // Lettura dati di input
   //
   int n ;

   cout << "Introdurre la dimensione del vettore: "; // Stampa la frase come un printf()
   cin >> n;

   double *pointer_vector_a ;
   double *pointer_vector_b ;


   // Alloca vettori
   //
   pointer_vector_a = new double[n] ;
   pointer_vector_b = new double[n] ;


   // Inizializza vettori
   //
   for(int i=0; i<n; i++) {
      pointer_vector_a[i] = 100.0 + i ;
```

```
      pointer_vector_b[i] = i ;
   }


   // Stampa vettori
   //
   cout << "Stampa vettore \n"
   for(int i=0; i<n; i++) {
      cout << " contenuto di pointer_vector_a[i] = " << pointer_vector_a[i] << "\n"  ;
   }

   cout << "Stampa vettore \n"
   for(int i=0; i<n; i++) {
      cout << " contenuto di pointer_vector_b[i] = " << pointer_vector_b[i] << "\n"  ;
   }


   // Scambia vettori
   //
   double *pointer_tmp ;

   pointer_tmp = pointer_vector_a ;
   pointer_vector_a = pointer_vector_b ;
   pointer_vector_b = pointer_tmp ;


   // Stampa vettori
   //
   cout << "Stampa vettore \n"
   for(int i=0; i<n; i++) {
      cout << " contenuto di pointer_vector_a[i] = " << pointer_vector_a[i] << "\n"  ;
   }

   cout << "Stampa vettore \n"
   for(int i=0; i<n; i++) {
      cout << " contenuto di pointer_vector_b[i] = " << pointer_vector_b[i] << "\n"  ;
   }



   // Cancella vettori
   //
   delete pointer_vector_a ;
   delete pointer_vector_b ;

}


// ------------------ End esempio_11 -----------------
```

Commenti

(1) Difetti del codice: il programma e' stato scritto con un approccio top down in un blocco unico, senza spezzare il problema in parti più semplici. Si noti come alcune parti del codice siano ripetute (la parte relativa alla stampa del vettore ma anche la parte di inizializzazione, seppur in misura minore). Il codice appare a prima vista poco chiaro (nonostante l'estrema semplicità del compito svolto dal programma) e di scarsa leggibilità. Inoltre il codice non può essere riutilizzato. Se in un altro contesto è necessario stampare il contenuto del vettore sarà necessario riscrivere completamente la parte relativa alla stampa.

Una soluzione rispetto alle critiche dell'esempio precedente consiste nell'utilizzare la tecnica della programmazione procedurale. In tal caso il task complesso viene spezzato in una serie di task più semplici (lettura della dimensione dei vettori, inizializzazione dei vettori, stampa dei vettori ecc.).

5.2.11.c Programma d'esempio sullo swap di due vettori II

- Nome file: main.cpp
- Contenuto: swap tra due vettori

main.cpp

```
// ------------------ Begin esempio_12 ---------------
//
// Swap tra due vettori
//

#include <iostream>


// Dichiarazione funzioni
//
int Inizializza_vettore(double *pointer_vector, int dim, double valore_iniziale) ;
int Stampa_vettore(double *pointer_vector, int dim) ;
int Leggi_intero(void) ;


using namespace std;

int main(void) {


   // Lettura dati di input
   //
   int n ;

   n = Leggi_intero() ;


   double *pointer_vector_a ;
   double *pointer_vector_b ;


   // Alloca vettori
   //
   pointer_vector_a = new double[n] ;
```

```
   pointer_vector_b = new double[n] ;


   // Inizializza vettori
   //
   Inizializza_vettore(pointer_vector_a, n, 100) ;
   Inizializza_vettore(pointer_vector_b, n, 0) ;


   // Stampa vettori
   //
   Stampa_vettore(pointer_vector_a, n) ;
   Stampa_vettore(pointer_vector_b, n) ;


   // Scambia vettori
   //
   double *pointer_tmp ;

   pointer_tmp = pointer_vector_a ;
   pointer_vector_a = pointer_vector_b ;
   pointer_vector_b = pointer_tmp ;


   // Stampa vettori
   //
   Stampa_vettore(pointer_vector_a, n) ;
   Stampa_vettore(pointer_vector_b, n) ;


   // Cancella vettori
   //
   delete pointer_vector_a ;
   delete pointer_vector_b ;

}


int Leggi_intero(void) {

   int n ;

   cout << "Introdurre la dimensione del vettore: "; // Stampa la frase come un printf()
   cin >> n;

   return n ;
}

int Inizializza_vettore(double *pointer_vector, int dim, double valore_iniziale) {
```

```
   for(int i=0; i<dim; i++) {
      pointer_vector[i] = valore_iniziale + i ;
   }

   return 0 ;
}

int Stampa_vettore(double *pointer_vector, int dim) {

   cout << "Stampa vettore \n" ;
   for(int i=0; i<dim; i++) {
      cout << " contenuto di pointer_vector[i] = " << pointer_vector[i] << "\n"  ;
   }

   return 0 ;
}


// ------------------ End esempio_12 -----------------
```

Commenti

(1) Difetti del codice: le funzioni sono separate dai dati. Come conseguenza la funzione non sa se il programmatore la chiamerà nella maniera corretta (ad es. se dim rappresenta effettivamente la corretta dimensione del vettore o ancora se pointer_vector è stato correttamente allocato), rendendo il codice poco sicuro.
Al solito, per questioni di chiarezza, è bene enucleare l'implementazione delle funzioni rispetto al file contenente il main.

5.2.12 Tipi derivati: le strutture

Le strutture sono un altro esempio di tipo derivato. Tramite le strutture il C consente di creare nuovi tipi a partire da tipi di variabili già esistenti. In sintesi una struttura non è altro che un insieme di tipi già definiti.
Questo consente di estendere il linguaggio in modo da gestire situazioni nuove. Se ad esempio abbiamo necessità di definire una nuova struttura di dati per affrontare un certo problema, il linguaggio ci permette di farlo.
Un esempio tipico potrebbe essere quello dei numeri complessi. Tipicamente i lunguaggi di programmazione classici non hanno tra i propri tipi di variabili il concetto di numero complesso. Il C tramite le strutture consente di creare un nuovo tipo variabile, dato da una coppia di numeri double, in grado di gestore / rappresentare un numero complesso.

5.2.12.a Programma: complex_structure

main.cpp

- Nome file: main.cpp
- Contenuto: main con manipolazione di alcuni numeri complessi

complex.hpp

- Nome file: complex.hpp
- Contenuto: definizione della strutttura complex

complex.cpp

- Nome file: complex.cpp
- Contenuto: implementazione delle funzioni per la manipolazione delle strutture di tipo complex.

5.2.12.b Commenti

Il difetto principale delle strutture sta nel fatto che queste consentono di "assemblare" / impacchettare solo dati, non è possibile inserire nelle strutture funzioni e/o operatori. Questa forte limitazione risulta particolaremente

evidente nel caso si sia creata una struttura per rappresentare e manipolare numeri complessi. Una volta definita la struttura "complex_number", data dalla combinazione di due numeri di tipo double, sarà possibile rappresentare un numero complesso ma non sarà possibile utilizzare i normali operatori $+ - */$ per eseguire le classiche operazioni aritmetiche su tali numeri. L'unica alternativa percorribile è quella riportata nell'esempio, in cui cioè si definiscono delle funzioni per gestire somme e prodotti di numeri complessi. Appare però evidente nel main come l'uso di tali funzioni appaia poco naturale e poco intuitivo.
Risulta quindi evidente come le strutture non rappresentino una reale estensione dei tipi di un linguaggio, in quanto tale costrutto si limita puramente a definire insiemi di tipi senza però permettere una reale definizione o ridefinizione ad esempio degli operatori che agiscono su tali nuove variabili. Quello che invece si vorrebbe per una piena estensibilità di un linguaggio sarebbe la possibilità di definire nuovi tipi variabili, definendo anche gli operatori che operano su questi tipi. Questo è esattamente quello che un linguaggio ad oggetti permette di raggiungere.
Vedremo successivamente come sarà possibile ottenere una piena estensione nella gestione dei numeri complessi tramite le classi del C++.

Un altro esempio di utilizzo del concetto di struttura è riportato di seguito.
Con questo esempio vengono illustrati due punti:

(1) Come sia possibile utilizzare / definire una struttura per trattare un polinomio.

(2) Come sia possibile trattare le operazioni tra polinomi in un'ottica procedurale.

Per entrambi i punti verranno discussi i principali difetti e limiti insiti in questa soluzione.

5.2.12.c Programma: polinomio_structure

main.cpp

- Nome file: main.cpp
- Contenuto: main con manipolazione di polinomi

polinomio.hpp

- Nome file: polinomio.hpp
- Contenuto: definizione della strutttura polinomio

polinomio.cpp

- Nome file: polinomio.cpp
- Contenuto: implementazione delle funzioni per la manipolazione delle strutture di tipo polinomio.

5.2.12.d Commenti

I principali vantaggi di questa soluzione sono i seguenti:

(1) La struttura permette di inglobare in un unico costrutto i due dati principali che caratterizzano il polinomio: (i) il suo grado (n) e (ii) il set di coefficienti (rappresentato in C da un vettore di dimensione $n+1$). In questo modo si ottiene un primo impacchettamento di dati in un'unica struttura. Questo permette di passare più facilmente i dati alle funzioni, in maniera più chiara, lineare e con minore rischi.

(2) La definizione delle funzioni: Prod_polinomi, Sum_polinomi ecc. tipiche di una programmazione procedurale, consente di utilizzare / riutilizzare facilmente queste funzioni all'interno del codice.

I limiti di questa impostazione, che verranno superati dalla programmazione ad oggetti, sono:

(1) La struttura Polinomio, benché consenta di inserire in un unico costrutto i dati che caratterizzano un polinomio, non permette di definire a tutti gli effetti un nuovo tipo di variabile. Non è infatti possibile scrivere:
Polinomio a, b, c; c = a + b ;

(2) Le funzioni per la manipolazione dei polinomi, sono separati dai dati. Se ad es. si cambia all'interno della struttura la modalità di gestione dei dati sarà necessario ridefinire anche le funzioni, pena la generazione di errori. Ad es. se si invertisse l'ordine dei coefficienti, le funzioni somma e prodotto continuerebbero a funzionare correttamente ma non la funzione Print_polinomio o una eventuale funzione che trovasse le radici dei polinomi di secondo grado, fornirebbe risultati errati. Poiché dati e funzioni sono separati, nulla garantisce che la coerenza tra questi due aspetti venga mantenuta da eventuali modifiche future.
In definitiva il punto debole di questo tipo di programmazione è la separazione tra dati e funzioni.

(3) L'utilizzo della struttura e delle funzioni, richiede comunque una certa conoscenza delle modalità seguite per l'implementazione. L'utente che utilizzasse questa struttura dovrebbe ad esempio sapere con quale criterio vengono inseriti i coefficienti del polinomio nel vettore all'interno della struttura (dal grado più basso a quello più alto? il contrario? una scelta diversa?). Questo rende l'uso della struttura più difficile, meno trasperente e potenzialmente pericoloso.
In questo caso il punto debole è la mancanza di mascheramento sulle modalità implementative interne della struttura.
A titolo di esempio: cosa accadrebbe se dopo aver allocato una struttura polinomio tramite la funzione Create_struct_polinomio con n=5, un utente dall'esterno ponesse successivamente il valore di dim a 10?

5.2.13 Overloading

L'overloading di una funzione indica la possibilità di definire funzioni con lo stesso nome ma argomenti diversi. Si veda l'esempio riportato sotto:

5.2.13.a Programma: overloading di funzioni

- Nome file: main.cpp
- Contenuto: overloading di funzioni

main.cpp

```
// ------------------ Begin esempio_13 ---------------
//
// Overloading di una funzione


#include <cstdlib>
#include <iostream>

int Max(int a, int b) ;
double Max(double a, double b) ;

using namespace std;

int main(void) {

   int ia = 2 ;
   int ib = 3 ;

   double da = 10.2 ;
   double db = 20.3 ;


   // Visualizzazione risultato
   //
   cout << "Il primo max e' " << Max(ia, ib) << " il secondo max e' "<< Max(da, db) << " \

}



int Max(int ia, int ib) {
   if(ia>ib) {
      return ia ;
   }
```

```
   else {
      return ib ;
   }
}

double Max(double da, double db) {
   if(da>db) {
      return da ;
   }
   else {
      return db ;
   }
}


// ------------------ End esempio_13 -----------------
```

5.2.14 Compilazione

La creazione di un eseguibile in C++ avviene in 3 passaggi:

1) precompilazione (direttive precedute da # nei file .cpp)

2) compilazione dei file .cpp e creazione di un codice oggetto .o (uno per ciascun file cpp)
   g++.exe -c main.cpp -o main.o -I"path file include"

3) linking dei vari moduli oggetto in un eseguibile
   g++.exe main.o -o prova.exe -L"path librerie" -lmc
   la flag -l indica l'utilizzo della libreria specificata di seguito, nell'esempio la libreria libmc.a (come si sarà notato, si omette nel nome della libreria che segue il comando "-l" il prefisso lib ed il suffisso ".a").

Sono utili i software per la gestione visuale dei progetti. Ad esempio sotto windows un interessante programma freeware è: dev-cpp for windows (http://www.bloodshed.net/dev/).
Tale ambiente permette in maniera molto semplice ed intuitiva di:

1) creare un progetto

2) precompilare e compilare i singoli moduli (file .cpp)

3) effettuare l'operazione di linking in un eseguibile finale

È buona norma separare l'implementazione delle funzioni dal file contentente il main. In generale sarebbe utile separare l'implementazione delle varie funzioni in diversi file, cercando di assegnare a ciascun file tutte le funzioni che sono fra loro imparentate.

5.2.15 Gestione della memoria in C/C++

Gestione della memoria

1) Code Segment: è l'area di memoria dove risiede il codice dell'applicazione.

2) Data Segment: questa area di memoria contiene le variabili globali o statiche definite nel codice.

3) stack: è l'area di memoria gestita in automatico dal compilatore. È in questa area che vengono allocate la variabili locali definite nelle varia funzioni. È il sistema operativo che si preoccupa di gestire tale area di memoria, allocando e deallocando lo spazio necessario quando le variabili vengono definite o escono dal loro scopo (al termine ad es. di una funzione).

4) heap (detta anche free store): questa è l'area di memoria gestita dinamicamente dall'utente. Ad esempio è in questa area dove viene allocato lo spazio per gestire gli array dinamici.

5.3 Programmazione ad oggetti

Perché un linguaggio si possa definire ad oggetti, deve possedere le seguenti proprietà:

- Incapsulamento e mascheramento
- Polimorfismo (overloading)
- Ereditarieta'

L'incapsulamento dei dati consiste nella possibilità di definire un costrutto unico (classe) in cui vengono incapsulati sia dati che funzioni. In questo modo si evita il problema presente nella programmazione procedurale in cui dati e funzioni viaggiano separatamente, con la conseguenza che non vi è mai la certezza che i dati inviati ad una funzione siano coerenti con essa.
Il mascheramento riguarda il fatto che certi dati siano nascosti all'utilizzatore della classe. Questo perché non vogliamo che l'utente possa alterare tali dati senza che la classe ne sia all'oscuro. Tipicamente i dati (variabili) presenti in un oggetto sono sempre mascherati, questo per evitare manipolazioni incontrollate da parte dell'utente. Verranno poi implementati nell'oggetto dei metodi che consentono di accedere (o meno) ai dati ed eventualmente di modificarli (in maniera però controllata).

5.3.1 Definizione di una classe

Di seguito è riportato a titolo di esempio la definizione di una classe in C++ con cui si definisce il concetto di quadrato in un piano.
Per implementare questa classe dobbiamo prima rispondere alle seguenti domande:

- Cosa caratterizza il concetto di quadrato?
- Quali sono le sue proprietà fondamentali?

La risposta a queste domande è:

- il quadrato è caratterizzato dall'essere un poligono regolare di quattro lati, quindi identici (pari ad l), con i quattro angoli interni pari a $\pi/2$. Il concetto di lato verrà sintetizzato tramite un dato di tipo double. Insito nel concetto di lato vi è poi il fatto che il numero che ne rappresenta la misura sia sempre maggiore o uguale a zero. Ovvero il double che rappresenterà il lato dovrà sempre essere positivo o al più nullo. Per l'angolo interno sarà sufficiente definire una funzione che restituisca $\pi/2$.
- Altra carattieristica del quadrato è quella di avere un perimetro ($4l$) ed un'area (l^2). Entrambe queste caratteristiche sono definite tramite una funzione (calcola perimetro e calcola area).

Da questa analisi discende l'implentazione riportata nel paragrafo successivo.

5.3.1.a Programma d'esempio sulla definizione di una classe (vedi codice libreria 03.01.01)

- Contenuto: esempio di definizione di una classe

quadrato.hpp

```
//
// Dichiarazione di una classe


#ifndef _QUADRATO_HPP
#define _QUADRATO_HPP

#define PI_GRECO 3.1415926

class Quadrato {

   private:

      double lato ;

   protected:

   public:

   // Costruttore
   //
   Quadrato(void) {  // in line
      lato = 0 ;
   };

   Quadrato(double _lato) {  // in line
      Set_lato(_lato) ;
   } ;

   // Distruttore
   //
   ~Quadrato() ;

   // Funzioni di input/output
   //
   double Get_lato(int indice_lato) {  // in line
      // indice_lato compreso tra 0 e 3
      //
      return lato ;
   } ;
```

```
   int Set_lato(double _lato) ;

   int Get_numero_lati(void) {  // in line
      return 4 ;
   } ;

   int Get_numero_angoli_interni(void) {  // in line
      return 4 ;
   } ;

   int Get_angolo_interno(int indice_angolo) {  // in line
      // indice_angolo compreso tra 0 e 3
      //
      return PI_GRECO/2 ;
   } ;

   double Get_area(void) {  // in line
      return lato * lato ;
   } ;

   double Get_perimetro(void) {  // in line
      return 4 * lato ;
   } ;

} ;

#endif
```

Commenti

(1) La variabile "lato" è stata dichiarata all'interno della sezione "private" ed è quindi accessibile solo all'interno della classe. Non può essere modificata da un utente esterno (vedi esempio successivo). Tale proprietà va sotto il nome di mascheramento dei dati.

(2) L'inizializzazione ovvero la costruzione di un oggetto di tipo Quadrato avviene tramite il costruttore (nell'esempio ne sono riportati due).

(3) La deallocazione di un oggetto di tipo Quadrato avviene tramite il distruttore (ne può essere definito solamente uno!).

(4) Vi sono poi due funzioni di input/output per gestire la modifica di "lato". Una funzione per ottenere gli angoli interni ed infine vi sono due funzioni che permettono di calcolare due proprietà fondamentali di questo oggetto: il perimetro e l'area.

(5) Si noti come alcune funzioni siano implementate direttamente nella dichiarazione della classe. Questo significa che le funzioni sono di tipo "in line".

L'interfaccia della classe è definita dall'insieme dei suoi metodi, quindi:

- i due costruttori:

```
Quadrato(void) ;
Quadrato(double lato_init) ;
```

- il distruttore:

```
~Quadrato() ;
```

- tre funzioni di input/output per il setting dei parametri:

```
int Set_lato(double lato_init) ;
double Get_lato(int indice_lato) ;
int Get_numero_lati(void) ;
int Get_angolo_interno(int indice_angolo) ;
int Get_numero_angoli_interni(void) ;
```

- due funzioni di calcolo:

```
double Get_area(void) ;
double Get_perimetro(void) ;
```

per un totale di otto metodi (vedi anche paragrafo successivo).
L'insieme di questi metodi è l'unico punto di congiunzione / interazione tra la classe ed il mondo.
Questo implica che qualunque modifica apportata all'implementazione di questi metodi, ovvero qualunque modifica eseguita all'interno della classe che lasci inalterate le dichiarazioni dei metodi (ovvero dell'interfaccia) non comporterà alcuna necessità di modifica al codice che utilizza tale classe.

quadrato.cpp

L'implementazione delle funzioni presenti nella classe Quadrato è effettuata nel corrispondente file .cpp:

```
//
// Utilizzo di una funzione


#include <cstdlib>
#include <iostream>

#include "quadrato.hpp"


using namespace std;



// ----------> Class: Quadrato
// ----------> Function name: ~Quadrato
// ----------> Description: Distruttore di Quadrato
//
Quadrato::~Quadrato(void) {
   cout << "Distruggo l'oggetto di tipo Quadrato \n" ;
}


// ----------> Class: Quadrato
// ----------> Function name: Set_lato
// ----------> Description: Setting del lato
//
int Quadrato::Set_lato(double _lato) {

   if(_lato>=0) {
      lato = _lato ;
      return 0 ;
   }
   else {
      cout << "Warning in Quadrato::Set_lato, _lato = " << _lato << " e' negativo \n" ;
      cout << "Nessun setting effettuato \n" ;
      return 1 ;
   }

}
```

main.cpp

Un esempio di main che utilizza la classe è:

```
// ------------------ Begin esempio_14 ---------------
//
// Utilizzo di una funzione
```

```
#include <cstdlib>
#include <iostream>

#include "quadrato.hpp"


using namespace std;

int main(void) {

   double lato ;

   // Legge il valore e lo copia in n come scanf()
   //
   cout << "Inserire lato del quadrato \n" ;
   cin >> lato;


   // Creo un oggetto (istanza) della classe Quadrato
   //
   Quadrato quadrato(lato) ;

   // Visualizzazione risultato
   //
   cout << "L'area del quadrato di lato " << lato << " e': " << quadrato.Get_area() << "\n


   quadrato.Set_lato(20.0) ;

   // Visualizzazione risultato
   //
   cout << "L'area del quadrato di lato " << quadrato.Get_lato() << " e': " << quadrato.Ge

}


// ------------------ End esempio_14 -----------------
```

L'output ottenuto a video è il seguente:

```
Inserire lato del quadrato
10
L'area del quadrato di lato 10 e': 100
L'area del quadrato di lato 20 e': 400
Distruggo l'oggetto di tipo Quadrato
```

Commenti

(1) Il mascheramento dei dati in questo caso consiste nel fatto che la variabile "lato" (dichiarata private) contenuta in un oggetto di tipo Quadrato risulta inaccessibile dall'esterno (ad es. dalla funzione main). L'unica maniera che l'utente ha a disposizione per alterare il contenuto della variabile "lato" è quella di utilizzare l'opportuna funzione di interfaccia Quadrato::Set_lato.
Anche la semplice interrogazione del contenuto di lato deve necessariamente passare attraverso l'utilizzo dell'altra funzione di interfaccia Quadrato::Get_lato.
In altri termini la variabile Quadrato::lato risulta "mascherata" ovvero protetta contro le manipolazioni esterne dell'utente ed è visibile unicamente all'interno (ovvero nello scopo) della classe Quadrato. Questo obiettivo è stato realizzato grazie al fatto di aver dichiarato la variabile "lato" all'interno del blocco "private".

5.3.2 Interfaccia di una classe

Una classe contiene dati e funzioni (chiamati anche metodi o operazioni). Tipicamente i dati di una classe non sono mai accedibili dall'esterno, ovvero vengono dichiarati privati o protetti. Come conseguenza, l'unico modo possibile per interagire con un oggetto è attraverso i suoi metodi, le sue operazioni. Un'operazione, come tutte le funzioni, è definita dal suo nome, da quali variabili prende in ingresso e da quale risultato restituisce. L'insieme di tutte le operazioni/metodi di un oggetto ne definisce l'interfaccia.
Si può quindi affermare che, esternamente, l'unica cosa che conosciamo di un oggetto è la sua interfaccia; essa è la sola via che abbiamo per relazionarci con l'oggetto stesso. Non sappiamo invece nulla di come l'interfaccia sia implementata all'interno dell'oggetto, ovvero non sappiamo nulla di come l'oggetto sia costruito al proprio interno. Potrebbe accadere che due oggetti abbiano la stessa interfaccia, ma questa sia implementata in modi differenti nei due oggetti. Questo significa che i due oggetti possono ricevere gli stessi messaggi dall'esterno ma reagire in modi diversi.

Il concetto di inferfaccia è un punto cardine della programmazione ad oggetti ed è alla base della riutilizzabilità del codice tipico di questo stile di programmmazione. Poiché un oggetto si relaziona con il mondo circostante solo tramite la sua interfaccia, qualunque modifica o miglioria alle sue procedure interne non impatterà l'uso che ne viene fatto da parte di tutti i programmi che lo utilizzano. Sarà anche possibile sostituire un oggetto con un altro avente la stessa interfaccia ma differente implementazione, senza dover modificare in alcun punto il codice generale che lo utilizza (polimorfismo). Questo consente di rendere estremamente modulare la gestione dei programmi e delle librerie, permettendo una riutilizzabilità del codice ad un livello superiore rispetto a quello ottenibile con lo stile procedurale.

5.3.3 Ereditarietà di classe

L'ereditarietà di classe consente in C++ di ottenere una nuova classe a partire da una classe pre-esistente, minimizzando così la quantità di codice da scrivere.
Di seguito riportiamo un semplice esempio, al fine di mostrare la sintassi.

5.3.3.a Esempio di base (vedi codice libreria 03.02.01)

main.cpp

- Nome file: main.cpp
- Contenuto: classe padre, classe derivata e main.

```
// ------------------ Begin esempio classe derivata ---------------
//
//


#include <cstdlib>
#include <iostream>



using namespace std;


class Classe_base {

   private:

      double x ;

   public:
      Classe_base(double _x) {
         cout << "Costruttore classe base \n" ;
         Set_x(_x) ;
      } ;

      ~Classe_base(void) {
         cout << "Distruttore classe base \n" ;
      } ;

      void Set_x(double _x) {
         x = _x ;
```

```
      } ;

      double Get_x(void) {
         return x ;
      } ;

      void Print(void) {
         cout << "x = " << x << " \n" ;
      } ;

} ;


class Classe_derivata {

   private:

      int y ;

   public:
      Classe_derivata(double _x, int _y) : Classe_base(_x) {
         cout << "Costruttore classe derivata \n" ;
         Set_y(_y) ;
      } ;

      ~Classe_derivata(void) {
         cout << "Distruttore classe derivata \n" ;
      } ;

      void Set_y(int _y) {
         y = _y ;
      } ;

      int Get_y(void) {
         return y ;
      } ;

      void Print(void) {
         Classe_base::Print() ;
         cout << "y = " << y << " \n" ;
      } ;

} ;


int main(void) {

   Classe_derivata obj(10.0, 1) ;
```

```
   obj.Print() ;

   obj.Se_x(2.2) ;

   obj.Se_y(3) ;

   obj.Print() ;


   Classe_derivata *pointer_obj ;

   pointer_obj = new Classe_derivata(20.0, 2) ;

   pointer_obj->Print() ;

   delete pointer_obj ;

   return 0 ;
}


// ------------------ End esempio classe derivata -----------------
```

5.3.3.b Ereditarietà tra classi: un esempio "avanzato"

Come esempio più sofisticato di ereditarietà tra classi, consideriamo il problema di rappresentare l'insieme dei poligoni (non necessariamente regolari) e all'interno di questo insieme considerare poi i quadrati ed i tringoli.
Da un punto di vista logico, il concetto di quadrato e di triangolo può essere ottenuto a partire da quello di poligono, applicando alcune restrizioni (o richiedendo il soddisfacimento di alcune proprietà aggiuntive). In particolare si considerino le seguenti definizioni:

- Poligono: un'area racchiusa all'interno di una linea spezzata chiusa (tipicamente non intrecciata).
  Caratteristiche del poligono:
  - possiede n lati
  - ogni lato ha una lunghezza maggiore di zero
  - possiede n angoli interni
  - un angolo è rappresentato da un valore compreso tra $(0, \pi)$.
- Quadrato: è un poligono con alcune proprietà aggiuntive.
  Caratteristiche del quadrato:

  - È un poligono
  - Ha quattro lati
  - I lati sono tutti uguali
  - Gli angoli interni sono uguali e pari a $\pi/2$.

- Triangolo isoscele: è un poligono con alcune proprietà aggiuntive. Caratteristiche del triangolo isoscele:

  - È un poligono
  - Ha tre lati
  - Due lati sono uguali

La programmazione ad oggetti consente di dare una rappresentazione fedele di questo mondo (formato da tre tipologie di elementi: i poligoni, i quadrati ed i triangoli isoscele), ottemperando a tre fondamentali requisiti:

- Il linguaggio deve consentire di minimizzare la quantità di codice necessario per descrivere le singole tipologie di oggetti. In altri termini l'aggiunta di una nuova tipologia di oggetti appartenente al "mondo", deve avvenire nella maniera più rapida, semplice e parsimoniosa possibile. (Riutilizzabilità del codice).

- La definizione di un tipo oggetto evoluto (ad es. un quadrato) andrebbe effettuata in termini di un oggetto più elementare (un generico poligono), consentendo una maggiore purezza e semplicità logica. (Chiarezza nel codice).

- La scrittura di codice per la manipolazione ad es. di poligoni non dovrebbe risentire dell'aggiunta di una nuova particolare tipologia di poligoni. Ovvero il codice deve risultare perfettamente modulare. (Uso del polimorfismo).

La soluzione resa disponibile dalla programmazione ad oggetti è quella dell'ereditarietà di classe.

In particolare di seguito è riportato un esempio di implementazione per il mondo dei poligoni.

5.3.3.c Programma: poligoni, quadrati e triangoli (vedi codice libreria 03.02.02)

main.cpp

- Nome file: main.cpp
- Contenuto: main per la manipolazione di poligoni, quadrati e triangoli isosceli.

```
//
// Esempio di derivazione tra classi e polimorfismo


#include <cstdlib>
#include <iostream>

#include "figura_triangolo_isoscele.hpp"
#include "figura_quadrato.hpp"

#include "funzione_stampa_perimetro_e_area.hpp"
#include "funzione_leggi_da_tastiera.hpp"


using namespace std;



int main(void) {

   double lato ;
   Figura_poligono *poligono ;

   // Lettura dati di input
   //
   Leggi_da_tastiera(&lato) ;


   // Creo un quadrato ed un triangolo
   //
   Figura_quadrato            quadrato(lato) ;
   Figura_triangolo_isoscele  triangolo_isoscele(lato) ;

   Stampa_perimetro_e_area(&quadrato) ;
   Stampa_perimetro_e_area(&triangolo_isoscele) ;


}
```

funzione_stampa_perimetro_e_area.hpp

- Nome file: funzione_stampa_perimetro_e_area.hpp

- Contenuto: Dichiarazione funzione per la stampa del perimetro/area di un poligono.

```
// ------------------ Begin esempio ---------------
//
//


#ifndef FUNZIONE_STAMPA_PERIMETRO_E_AREA_HPP
#define FUNZIONE_STAMPA_PERIMETRO_E_AREA_HPP

int Stampa_perimetro_e_area(Figura_poligono *poligono) ;

#endif


// ------------------ End esempio -----------------
```

funzione_stampa_perimetro_e_area.cpp

- Nome file: funzione_stampa_perimetro_e_area.cpp
- Contenuto: Implementazione funzione per la stampa del perimetro/area di un poligono.

```
// ------------------ Begin esempio ---------------
//
// Utilizzo di una funzione


#include <cstdlib>
#include <iostream>

#include "figura_poligono.hpp"


using namespace std;



int Stampa_perimetro_e_area(Figura_poligono *poligono) {

   if(poligono != NULL) {
      // Visualizzazione risultato
      //
      cout << "poligono di " << poligono->Get_numero_lati() << " lati \n" ;
      cout << "   Area      = " << poligono->Get_area() << " \n" ;
      cout << "   perimetro = " << poligono->Get_perimetro() << " \n" ;
```

```
      return 0 ;
   }
   else {
      return 1 ;
   }
}


// ------------------ End esempio -----------------
```

figura_poligono.hpp

- Nome file: figura_poligono.hpp
- Contenuto: definizione della classe Poligono

```
//
// Dichiarazione di una classe


#ifndef _FIGURA_POLIGONO_HPP
#define _FIGURA_POLIGONO_HPP


#include <cstdlib>
#include <iostream>

#define PI_GRECO 3.1415926

using namespace std;



class Figura_poligono {

   private:

      int numero_lati ;

      double *lati ;
      double *angoli_interni ;


   protected:

      int Allocazione_iniziale(int _numero_lati) ;

      // costruttore con solo allocazione della memoria
```

```
   //
   Figura_poligono(int _numero_lati) {
      Allocazione_iniziale(_numero_lati) ;
   } ;

public:

   // Costruttori
   //

   // costruttore di default
   //
   Figura_poligono(void) {  // in line
      Allocazione_iniziale(0) ;
   };

   // costruttore generico
   //
   Figura_poligono(int _numero_lati, double *_lati, double *_angoli_interni) {
      Allocazione_iniziale(_numero_lati) ;

      for(int i=0; i<_numero_lati; i++) {
         Set_lato(i, _lati[i]) ;
         Set_angolo_interno(i, _angoli_interni[i]) ;
      }
   } ;

   // costruttore di un poligono regolare
   //
   Figura_poligono(int _numero_lati, double _lato) {
      Allocazione_iniziale(_numero_lati) ;

      for(int i=0; i<_numero_lati; i++) {
         Set_lato(i, _lato) ;
         Set_angolo_interno(i, PI_GRECO*(double(numero_lati)-2.0)/double(numero_lati))
      }
   } ;

   // Distruttore
   //
   ~Figura_poligono(void) {
      cout << "Distruggo l'oggetto di tipo Figura_poligono \n" ;
      delete lati ;
      delete angoli_interni ;
   } ;

   // Funzioni di input/output
   //
   double Get_lato(int indice_lato) {  // in line
```

```
         // indice_lato compreso tra 0 e numero_lati
         //
         if(indice_lato >=0 && indice_lato <numero_lati) {
            return lato[indice_lato] ;
         }
         else {
            return 0 ;
         }
      } ;

      double Get_angolo_interno(int indice_angolo) {  // in line
         // indice_lato compreso tra 0 e numero_lati
         //
         if(indice_angolo >=0 && indice_angolo <numero_lati) {
            return angoli_interni[indice_angolo] ;
         }
         else {
            return 0 ;
         }
      } ;

      int Get_numero_lati(void) {  // in line
         return numero_lati ;
      } ;

      int Set_lato(int indice_lato, double lato_singolo) ;

      int Set_angolo_interno(int indice_angolo, double angolo_interno) ;



      // Funzioni di calcolo
      //
      virtual double Get_area(void) = 0 ;

      virtual double Get_perimetro(void) {
         double perimetro = 0 ;

         for(int i=0; i<_numero_lati; i++) {
            perimetro = perimetro + Get_lato(i) ;
         }

         return perimetro ;
      };

} ;

#endif
```

figura_poligono.cpp

- Nome file: figura_poligono.cpp
- Contenuto: implementazione delle funzioni per la classe Poligono

```
//
// Utilizzo di una funzione


#include <cstdlib>
#include <iostream>

#include "figura_poligono.hpp"


using namespace std;



// ----------> Class: Figura_poligono
// ----------> Function name: Allocazione_iniziale
// ----------> Description: Setting del numero di lati e della memoria
//
int Figura_poligono::Allocazione_iniziale(int _numero_lati) {

   if(_numero_lati >0) {

      numero_lati = _numero_lati ;

      lati = new double[numero_lati] ;
      angoli_interni = new double[numero_lati] ;

      for(int i=0; i<numero_lati; i++) {
         lati[i] = 0 ;
         angoli_interni[i] = 0 ;
      }

      return 0 ;
   }
   else {

      numero_lati = 0 ;
      lati = NULL ;
      angoli_interni = NULL ;

      return 1 ;
   }
```

```
}


// ----------> Class: Figura_poligono
// ----------> Function name: Set_lato
// ----------> Description: Setting del lato
//
int Figura_poligono::Set_lato(int indice_lato, double lato_singolo) {

   if(indice_lato>=0 && indice_lato <Get_numero_lati()) {
      if(lato_singolo > 0) {
         lati[indice_lato] = lato_singolo ;
         return 0 ;
      }
      else {
         return 1 ;
      }
   }
   else {
      return 1 ;
   }

}


// ----------> Class: Figura_poligono
// ----------> Function name: Set_angolo_interno
// ----------> Description: Setting dell'angolo interno
//
int Figura_poligono::Set_angolo_interno(int indice_angolo, double angolo_interno) {

   if(indice_angolo>=0 && indice_angolo <Get_numero_lati()) {
      if(angolo_interno > 0) {
         angoli_interni[indice_angolo] = angolo_interno ;
         return 0 ;
      }
      else {
         return 1 ;
      }
   }
   else {
      return 1 ;
   }

}
```

figura_quadrato.hpp

- Nome file: figura_quadrato.hpp
- Contenuto: definizione della classe Quadrato

```
//
// Dichiarazione di una classe


#ifndef _FIGURA_QUADRATO_HPP
#define _FIGURA_QUADRATO_HPP


#include <cstdlib>
#include <iostream>

#include "figura_poligono.hpp"


using namespace std;



class Figura_quadrato : public Figura_poligono {

   private:

   protected:

   public:

   // Costruttore
   //
   Figura_quadrato(double _lato) : Figura_poligono(4) {  // 4 lati
      for(int i=0; i<Get_numero_lati(); i++) {
         Set_lato(i, _lato) ;  // lati uguali
         Set_angolo(i, PI/2) ; // angoli uguali
      }
   } ;

   // Distruttore
   //
   virtual ~Figura_quadrato(void) {
      cout << "Distruggo l'oggetto di tipo Figura_quadrato \n" ;
   } ;

   // Funzioni di input/output specifici dell'oggetto
   //
   double Get_lato(void) {  // in line
      return Get_lato(0) ;
   } ;
```

```
   // Funzioni di calcolo
   //
   virtual double Get_area(void) {  // in line
      return Get_lato() * Get_lato() ;
   } ;

} ;

#endif
```

figura_triangolo_isoscele.hpp

- Nome file: figura_triangolo_isoscele.hpp
- Contenuto: definizione della classe Triangolo

```
//
// Dichiarazione di una classe


#ifndef _FIGURA_TRIANGOLO_ISOSCELE_HPP
#define _FIGURA_TRIANGOLO_ISOSCELE_HPP


#include <cstdlib>
#include <iostream>
#include <math.h>

#include "figura_poligono.hpp"



using namespace std;


class Figura_triangolo_isoscele : public Figura_poligono {

   private:

   protected:

   public:

   // Costruttore
   //
   Figura_triangolo_isoscele(void) : Figura_poligono() {  // in line
```

```
   };

   Figura_triangolo_isoscele(double _lato_base,
                             double _lato_uguale,
                             double _angolo_unico) : Figura_poligono(3) {  // in line

      Set_lato(0, _lato_base) ;    // lato alla base
      Set_lato(1, _lato_uguale) ;  // lati uguali
      Set_lato(2, _lato_uguale) ;  // lati uguali
      Set_angolo(0, (PI-_angolo_unico)/2) ; // angoli uguali alla base
      Set_angolo(1, (PI-_angolo_unico)/2) ; // angoli uguali alla base
      Set_angolo(2, _angolo_unico) ;        // angolo diverso al vertice
   } ;

   // Distruttore
   //
   virtual ~Figura_triangolo_isoscele(void) {
      cout << "Distruggo l'oggetto di tipo Figura_triangolo_isoscele \n" ;
   } ;


   // Funzioni di calcolo specifiche della classe
   //
   double Get_altezza(void) {  // in line
      return sqrt( Get_lato_uguale()*Get_lato_uguale() - Get_base() * Get_base() / 4.0 )
   } ;


   // Funzioni di calcolo
   //
   virtual double Get_area(void) {  // in line
      return Get_base() * Get_altezza() / 2.0 ;
   } ;


} ;

#endif
```

5.3.3.d Commenti

I principali vantaggi di questa soluzione sono i seguenti:

(1) La classe padre (Poligono), è una classe generale per implementare il concetto di poligono. Si noti come tutte le proprietà elencate per il

concetto matematico di poligono trovano una corrispondenza precisa all'interno della classe.

(2) Le classi figlie (Quadrato e Triangolo_isoscele), sono "derivate" dalla classe Poligono. Il codice necessario a descriverle è ristretto al minimo ed implementa unicamente le proprietà aggiuntive che caratterizzano i quadrati o i triangoli_isosceli all'interno della famiglia generale dei poligoni.

(3) La classe padre (Poligono) è stata resa virtuale pura, questo perché le funzioni:

```
double Get_area(void) ;
double Get_perimetro(void) ;
```

sono dichiarate virtuali. Questo consente di scrivere del codice (il main nell'esempio riportato sopra) in grado di trattare qualunque oggetto del mondo dei poligoni senza preoccuparsi della sua implementazione interna o della sua tipologia (polimorfismo evoluto). L'aggiunta di una nuova tipologia di oggetto (ad es. i rettangoli o i pentagoni regolari), non necessità di modificare il codice della funzione:

```
int Stampa_caratteristiche_poligono(Poligono * oggetto_poligono) ;
```

Questo consente una vera e piena riutilizzabilità del codice.

Un secondo esempio di utilizzo dell'ereditarietà per creare una nuova classe a partire da una esistente è il seguente:

- Polinomio di primo grado: un polinomio della forma $a + b \cdot x$.
  Caratteristiche di un polinomio di primo grado:
  - il coefficiente del termine di grado zero (a).
  - il coefficiente del termine di grado uno (b).
  - possedere una radice ($x = -a/b$).
- Polinomio di secondo grado: un polinomio della forma $a + b \cdot x + c \cdot x^2$.
  Caratteristiche di un polinomio di secondo grado:
  - il coefficiente del termine di grado zero (a).
  - il coefficiente del termine di grado uno (b).
  - il coefficiente del termine di grado due (c).

– possedere due radici.

In questo caso le tipologie di oggetti del "mondo" sono due: (a) i polinomi di primo grado e (b) i polinomi di secondo grado.
Ovviamente è possibile definire due classi indipendenti che caratterizzano le due tipologie di oggetti. Risulta però più vantaggioso, sia in termini di chiarezza logica sia in termini di economicità nella struttura del codice, utilizzare la derivazione tra classi.

5.3.3.e Programma: Polinomi di primo e secondo grado tramite ereditarietà (vedi codice libreria 03.02.03)

main.cpp

- Nome file: main.cpp
- Contenuto: main per la manipolazione di polinomi di primo e secondo grado.

```
#include <cstdlib>
#include <iostream>

#include "polinomio_primo_grado.hpp"
#include "polinomio_secondo_grado.hpp"

using namespace std;

int main(int argc, char *argv[])
{
    double root_1_grado ;

    // Costruzione polinomio primo grado
    //
    Polinomio_primo_grado p_1_grado(1, 1) ;

    // Costruzione polinomio primo grado
    //
    Polinomio_secondo_grado p_2_grado(1, 1, 2) ;


    // Stampa delle info
    //
    p_1_grado.Print() ;

    p_2_grado.Print() ;
```

```
    p_1_grado.Get_root(0, root_1_grado) ;

    cout << "radice polinomio 1 grado " << root_1_grado << " \n" ;


    return 0 ;
}
```

polinomio_primo_grado.hpp

- Nome file: polinomio_primo_grado.hpp
- Contenuto: definizione della classe Polinomio_primo_grado

```
#ifndef _POLINOMIO_PRIMO_GRADO_HPP
#define _POLINOMIO_PRIMO_GRADO_HPP

#include <iostream>


using namespace std;


class Polinomio_primo_grado {

    private:
        // polinomio = coeff_grado_zero + coeff_grado_uno * x
        //
        double coeff_grado_zero ;
        double coeff_grado_uno ;

    public:


        Polinomio_primo_grado(void) : Punto() {
            Set_coeff_grado_zero(0) ;
            Set_coeff_grado_uno(0) ;
        } ;

        Polinomio_primo_grado(double _coeff_grado_zero, double _coeff_grado_uno) {
            Set_coeff_grado_zero(_coeff_grado_zero) ;
            Set_coeff_grado_uno(_coeff_grado_uno) ;
        } ;

        // Lo studente e' chiamato ad implementare il costruttore di
        // copia.
        //
        Polinomio_primo_grado( const Polinomio_primo_grado& c ) ;
```

```
        ~Polinomio_primo_grado(void) {
            cout << "invoco il distruttore di polinomio di 1 grado << "\n";
        } ;

        int Set_coeff_grado_zero(double _coeff_grado_zero) {
            coeff_grado_zero = _coeff_grado_zero ;
            return 0 ;
        } ;

        int Set_coeff_grado_uno(double _coeff_grado_uno) {
            coeff_grado_uno = _coeff_grado_uno ;
            return 0 ;
        } ;

        double Get_coeff_grado_zero(void) ;

        double Get_coeff_grado_uno(void) ;

        virtual void Print(void) ;

        // Fornisce le radici del polinomio di
        // primo grado (ovvero una sola radice!).
        // In questo caso root_counter potra'
        // assumere solo il valore 0 (corrispondente
        // alla prima ed unica radice del polinomio
        // di primo grado).
        // Il risultato sara' inserito nella variabile
        // root (passata tramite reference).
        //
        virtual int Get_root(int root_counter, double &root) ;

} ;

#endif
```

polinomio_secondo_grado.hpp

- Nome file: polinomio_secondo_grado.hpp
- Contenuto: definizione della classe Polinomio_secondo_grado

```
#ifndef _POLINOMIO_SECONDO_GRADO_HPP
```

```
#define _POLINOMIO_SECONDO_GRADO_HPP

#include <iostream>


using namespace std;

#include "polinomio_primo_grado.hpp"

class Polinomio_secondo_grado : Polinomio_primo_grado {

    private:
        // polinomio = coeff_grado_zero + coeff_grado_uno * x +
        //             + coeff_grado_due * x^2
        //
        double coeff_grado_due ;

    public:


        Polinomio_secondo_grado(void) : Polinomio_primo_grado() {
            Set_coeff_grado_due(0) ;
        } ;

        Polinomio_secondo_grado(
            double _coeff_grado_zero,
            double _coeff_grado_uno,
            double _coeff_grado_due
        ) : Polinomio_primo_grado(_coeff_grado_zero, _coeff_grado_uno) {

            Set_coeff_grado_due(_coeff_grado_due) ;
        } ;

        // Lo studente e' chiamato ad implementare il costruttore di
        // copia.
        //
        Polinomio_secondo_grado( const Polinomio_secondo_grado& obj ) ;


        ~Polinomio_secondo_grado(void) {
            cout << "invoco il distruttore di polinomio di 2 grado << "\n";
        } ;

        int Set_coeff_grado_due(double _coeff_grado_due) ;

        double Get_coeff_grado_due(void) ;

        virtual void Print(void) ;
```

```
        // Fornisce le radici del polinomio di
        // secondo grado (ovvero due soli radici!).
        // In questo caso root_counter potra'
        // assumere solo i valori 0 e 1 (corrispondente
        // alla prima e seconda radice del polinomio
        // di secondo grado).
        // Il risultato sara' inserito nella variabile
        // root (passata tramite reference).
        //
        virtual int Get_root(int root_counter, double &root) ;

} ;

#endif
```

Un terzo esempio di utilizzo dell'ereditarietà per creare una nuova classe a partire da un classe già esistente, è il seguente:

- Punto in un piano: un punto in un piano cartesiano è rappresentato tramite due coordinate (x, y).
  Caratteristiche del punto:
  - avere due coordinate che lo identifichino nel piano
- Cerchio: è una figura geometrica nel piano caratterizzata dall'insieme dei punti equidistanti rispetto ad un punto dato.
  Caratteristiche del cerchio:
  - È caratterizzato da un centro (un punto nel piano).
  - È definito da un raggio.
  - Le dimensioni del raggio sono positive.

In questo caso gli oggetti del mondo sono due: (a) il punto e (b) il cerchio.
Ovviamente è possibile definire due classi indipendenti che caratterizzano le due tipologie di oggetti. Risulta però più vantaggioso sia in termini di chiarezza logica sia in termini di economicità nella struttura del codice, utilizzare la derivazione tra classi.

5.3.3.f Programma: classe_cerchio tramite l'ereditarietà (vedi codice libreria 03.02.04)

main.cpp

- Nome file: main.cpp
- Contenuto: main per la manipolazione di cerchi

```
#include <cstdlib>
#include <iostream>

#include "punto.hpp"
#include "cerchio_per_derivaz.hpp"

using namespace std;

int main(int argc, char *argv[])
{

    // ************************************************************
    // **                   Costruzione oggetti                  **
    // ************************************************************

    Punto p(-1,-1) ;


    // ************************************************************
    // ** Costruzione di un cerchio tramite                      **
    // ** l'ereditarieta' di classe                              **
    // ************************************************************


    // Costruzione di un cerchio tramite
    // l'ereditarieta' di classe.
    // Si crea un cerchio di raggio 5 con centro in (1,1)
    //
    Cerchio_per_derivaz cerchio_1(5,1,1) ;



    // ************************************************************
    // **         Stampa delle info del cerchio                  **
    // ************************************************************

    cerchio_1.Print() ;
```

```
    // ********************************************************
    // **   Cambio del centro del cerchio                    **
    // ********************************************************

    // Cambio il centro di cerchio_1.
    // Posso usare direttamente  Set_coordinate
    //
    cerchio_1.Set_coordinate(-2,-2) ;



    // ********************************************************
    // **         Stampa delle info del cerchio              **
    // ********************************************************

    cerchio_1.Print() ;


    // ********************************************************
    // **                    End program                     **
    // ********************************************************

    return 0 ;

}
```

punto.hpp

- Nome file: punto.hpp
- Contenuto: definizione della classe Punto

```
#ifndef _PUNTO_HPP
#define _PUNTO_HPP

#include <iostream>


using namespace std;


class Punto {

    private:
        double x ;
        double y ;

    public:
```

```
    Punto(void) {
        x = 0 ;
        y  = 0 ;
    } ;

    Punto(double _x, double _y) {
        Set_coordinate(_x, _y) ;
    } ;

    Punto::Punto( const Punto& c ) {
        x = c.x;
        y = c.y;
    } ;


    ~Punto(void) {
        cout << "invoco il distruttore " << "\n";
    } ;


    int Set_coordinate(double _x, double _y) {
        x = _x ;
        y  = _y ;
        return 0 ;
    } ;


    void Print(void) {
        cout << "punto = [" << x << "," << y << "] \n";
    } ;

} ;

#endif
```

cerchio_tramite_ereditarieta.hpp

- Nome file: cerchio_tramite_ereditarieta.hpp
- Contenuto: definizione della classe Cerchio_tramite_ereditarieta

5.3.3.g Commenti

Di seguito alcuni commenti:

(1) Nell'esempio sull'uso dell'ereditarietà riguardante i poligono, avevamo due insiemi: (a) i poligono e (b) i quadrati. L'ereditarieà permetteva di definire i quadrati partendo dai poligono, ovvero applicava una "restrizione" all'insieme più generale per ottenere un insieme più ristretto.
Nell'esempio riguardante i polinomi di primo e secondo grado, avevamo due insiemi: (a) i polinomi di primo grado e (b) i polinomi di secondo grado. L'ereditarietà permetteva di definire i polinomi di secondo grado partendo da quelli di primo grado, ovvero applicava una estensione all'insieme più ristretto per ottenere un insieme più generale.
Nell'esempio corrente sui cerchi, abbiamo due insiemi: (a) i punti e (b) i cerchi. L'ereditarietà permette di definire i cerchi a partire dai punti. In questo caso i due insiemi non sono in un rapporto di inclusione l'uno rispetto all'altro (a meno di non vedere i punti come casi particolari di cerchi di raggio nullo).
Questo per osservare come l'ereditarietà sia estremamente versatile nell'uso.

5.3.4 Composizione di oggetti vs ereditarietà

Vi sono sostanzialmente due modi per estendere e definire nuovi oggetti a partire da oggetti già esistenti:

(1) Tramite l'ereditarietà tra classi.

(2) Tramite la composizione di oggetti in una nuova classe.

Modi ed esempi relativi alla prima metodologia sono stati dati nei paragrafi precedenti. Occupiamoci ora della tecnica di composizione.
La composizione è una modalità estremamente intuitiva: una nuova classe è ottenuta includendo come membri altri oggetti. In tal modo certi compiti che la nostra classe è chiamata a svolgere possono essere delegati ad alcuni oggetti da essa inclusi, o certe azioni complesse possono essere ottenute coordinando opportunamente gli oggetti in questione.
La tecnica di composizione è molto intuitiva in quanto è prassi corrente in qualunque attività umana, quella di assemblare insieme oggetti semplici per ottenere un oggetto più complesso.
Come esempio di composizione riprendiamo il caso del punto e del cerchio, cercando di definire una classe cerchio partendo dalla classe punto, senza ricorrere all'ereditarietà come fatto precedentemente.

5.3.4.a Programma: classe_cerchio tramite composizione (vedi codice libreria 03.03.01)

main.cpp

- Nome file: main.cpp
- Contenuto: main per la manipolazione di cerchi

```
#include <cstdlib>
#include <iostream>

#include "../Esempio_classe_cerchio/punto.hpp"
#include "../Esempio_classe_cerchio/cerchio_per_ereditarieta.hpp"
#include "cerchio_per_composizione_tramite_aggregazione.hpp"
#include "cerchio_per_composizione_tramite_cognizione.hpp"

using namespace std;

int main(int argc, char *argv[])
{

    // ******************************************************
```

```
// **                  Costruzione oggetti               **
// ********************************************************

// Costruzione di un punto
//
Punto p(-1,-1) ;

// Costruzione di un cerchio tramite l'ereditarieta' di classe
// cerchio di raggio 5 con centro in (1,1)
//
Cerchio_per_ereditarieta cerchio_1(5,1,1) ;



// ********************************************************
// ** Costruzione di cerchi tramite la composizione di    **
// ** oggetti: nello specifico un double (raggio) ed un   **
// ** punto.                                              **
// ********************************************************

// cerchio di raggio 5 con centro in (0,0)
// il cerchio in questo caso e' responsabile
// del punto che contiene.
//
Cerchio_per_composizione_tramite_aggregazione cerchio_2(5,0,0) ;


// cerchio di raggio 5 con centro in p
//
Cerchio_per_composizione_tramite_cognizione cerchio_3(5, &p) ;


// cerchio di raggio 10 con centro in p
//
Cerchio_per_composizione_tramite_cognizione cerchio_4(10, &p) ;



// ********************************************************
// **         Stampa delle info sui vari oggetti         **
// ********************************************************

cerchio_1.Print() ;

cerchio_2.Print() ;

cerchio_3.Print() ;

cerchio_4.Print() ;
```

```
    // ********************************************************
    // **   Cambio dei centri dei cerchi                     **
    // ********************************************************

    // Cambia solo il centro di cerchio_1.
    // Posso usare direttamente  Set_coordinate
    //
    cerchio_1.Set_coordinate(-2,-2) ;


    // Cambia solo il centro di cerchio_2.
    // Il cambio del centro passa attraverso la
    // delegare all'oggetto interno di tipo Punto
    //
    cerchio_2.Get_centro().Set_coordinate(3,3) ;


    // Cambia anche il centro di cerchio_4
    //
    cerchio_3.Get_centro()->Set_coordinate(2,2) ;



    // ********************************************************
    // **         Stampa delle info sui vari oggetti         **
    // ********************************************************

    cerchio_1.Print() ;

    cerchio_2.Print() ;

    cerchio_3.Print() ;

    cerchio_4.Print() ;


    // ********************************************************
    // **                    End program                     **
    // ********************************************************

    return 0 ;
}
```

cerchio_per_composizione_tramite_cognizione.hpp

- Nome file: cerchio_per_composizione_tramite_cognizione.hpp

- Contenuto: definizione della classe Cerchio_per_composizione_tramite_cognizione

```
#ifndef _CERCHIO_PER_COMPOSIZIONE_TRAMITE_COGNIZIONE_HPP
#define _CERCHIO_PER_COMPOSIZIONE_TRAMITE_COGNIZIONE_HPP

#include <iostream>


using namespace std;


class Cerchio_per_composizione_tramite_cognizione {

    private:
        Punto *pnt_centro ;
        double raggio ;

    public:

        // Costruttori
        //
        Cerchio_per_composizione_tramite_cognizione(void) {
            raggio = 0 ;
        } ;

        Cerchio_per_composizione_tramite_cognizione(double _raggio, Punto *_pnt_centro) {
            pnt_centro = _pnt_centro ;
            raggio = _raggio ;
        } ;

        Cerchio_per_composizione_tramite_cognizione( const Cerchio_per_composizione_tramit
            pnt_centro = c.pnt_centro ;
            raggio = c.raggio;
        } ;


        // Distruttore
        //
        ~Cerchio_per_composizione_tramite_cognizione(void) {
            cout << "invoco il distruttore di cerchio" << "\n";
        } ;


        // Funzioni di setting
        //
        int Set_raggio(double raggio_init) {
            raggio = raggio_init ;
            return 0 ;
        } ;
```

```
        // Funzioni di getting
        //
        double Get_raggio(void) {
            return raggio ;
        } ;

        Punto *Get_centro(void) {
            return pnt_centro ;
        } ;


        // Funzioni di printing
        //
        void Print(void) {
            pnt_centro->Print() ;
            cout << "raggio cerchio = " << raggio << " \n";
        } ;

} ;

#endif
```

cerchio_per_composizione_tramite_aggregazione.hpp

- Nome file: cerchio_per_composizione_tramite_aggregazione.hpp
- Contenuto: definizione della classe Cerchio_per_composizione_tramite_aggregazione

```
#ifndef _CERCHIO_PER_COMPOSIZIONE_TRAMITE_AGGREGAZIONE_HPP
#define _CERCHIO_PER_COMPOSIZIONE_TRAMITE_AGGREGAZIONE_HPP

#include <iostream>


using namespace std;


class Cerchio_per_composizione_tramite_aggregazione {

    private:
        Punto centro ;
        double raggio ;
```

```
public:


    // Costruttori
    //
    Cerchio_per_composizione_tramite_aggregazione(void) {
        raggio = 0 ;
    } ;

    Cerchio_per_composizione_tramite_aggregazione(double _raggio, double _x, double _y
        centro.Set_coordinate(_x, _y) ;
        raggio = _raggio ;
    } ;

    Cerchio_per_composizione_tramite_aggregazione(const Cerchio_per_composizione_trami
        centro(obj.Get_centro()) ;
        raggio = obj.raggio;
    } ;


    // Distruttore
    //
    ~Cerchio_per_composizione_tramite_aggregazione(void) {
        cout << "invoco il distruttore di cerchio" << "\n";
    } ;


    // Funzioni di setting
    //
    int Set_raggio(double _raggio) {
        raggio = _raggio ;
        return 0 ;
    } ;


    // Funzioni di getting
    //
    double Get_raggio(void) {
        return raggio ;
    } ;

    Punto &Get_centro(void) {
        return centro ;
    } ;


    // Funzioni di printing
    //
    void Print(void) {
```

```
            centro.Print() ;
            cout << "raggio cerchio = " << raggio << " \n";
        } ;

} ;

#endif
```

5.3.4.b Commenti

Di seguito alcuni commenti:

(1) Si noti le due possibili declinazione nell'uso della composizione di oggetti: (a) tramite aggregazione e (b) tramite cognizione.
Nel caso dell'aggregazione, l'oggetto Punto è contenuto fisicamente nella classe Cerchio, sarà quindi compito della classe Cerchio gestire l'oggetto Punto (allocazione e deallocazione). In questo caso ad esempio se si crea un oggetto Cerchio tramite aggregazione, l'oggetto Punto in esso contenuto, inizierà la propria esistenza con la nascita di cerchio e la terminerà con la sua fine. In altri termini l'oggetto Punto non ha un'esistenza autonoma rispetto al Cerchio.
Nel caso della cognizione invece si ha la situazione opposta: il Cerchio ha solo cognizione dell'oggetto Punto, la cui esistenza però è indipendente dal Cerchio. Se si alloca un oggetto Cerchio tramite cognizione, l'oggetto Punto a cui il cerchio si riferisce, pre-esiste a quest'ultimo e non cesserà necessariamente di esistere al momento della deallocazione del cerchio. Potrà anche capitare che l'oggetto Punto in questione venga sostituito da un altro oggetto dello stesso tipo. I due oggetti sono quindi in una relazione reciproca più debole della precedente.

(2) Ritornando alla questione delle tecniche esistenti per il riutilizzo del codice (ereditarietà e composizione), vi sono molto differenze rilevanti. Una delle più significative sta nel fatto che nella composizione, la classe complessa (quella che include gli oggetti al proprio interno), necessità unicamente di conoscere la sola interfaccia degli oggetti che vuole includere. I dettagli implementativi di questi ultimi sono invece irrilevanti. Questo da un lato ha come svantaggio che l'interfaccia degli oggetti va stabilita con grande cura e non più modificata, ma dall'altro consente un notevole riutilizzo nel codice. Infatti sarà possibile sostituire un oggetto contenuto con un altro, purché quest'ultimo implementi la stessa interfaccia.

In generale una delle regole fondamentali per il riutilizzo del codice nella programmazione ad oggetti è quella di ricorrere maggiormente alla composizione di oggetti rispetto all'ereditarietà, di cui spesso si fa abuso.

5.3.5 Ridefinizione degli operatori

Come esempio di ridefinizione degli operatori in una classe, ritorniamo sull'esempio dei numeri complessi. Avevamo visto come nell'ambito della programmazione procedurale in C fosse possibile introdurre una rappresentazione dei numeri complessi tramite il concetto di struttura. Avevamo anche esaminato i limiti di questo approccio, fondamentalmente riconducibile all'impossibilità di una piena rappresentazione dei numeri complessi tramite un nuovo tipo. Tali limiti sono perfettamente superati e risolti nell'ambito della programmazione ad oggetti.
Di seguito viene proposta la riscrittura del codice basato su strutture, in termini di classi.

5.3.5.a Programma: complex_class (vedi codice libreria 03.04.01)

main.cpp

- Nome file: main.cpp
- Contenuto: main con manipolazione di alcuni numeri complessi

```
#include <cstdlib>
#include <iostream>
#include "complex.hpp"
using namespace std;

int main(int argc, char *argv[])
{
    Complex a(1,1) ; // viene invocato il costruttore double, double
    Complex b; // viene invocato il costruttore di default
    Complex c; // viene invocato il costruttore di default

    c = b = a ; // viene invocato l'operatore = due volte
    c= a+b ;    // viene invocato l'operatore +
    c.Print() ;


    Complex x ; // viene invocato il costruttore di default
    x = a ;     // viene invocato l'operatore =

    Complex y = a ; // viene invocato il cpy constructor


    Test_copy_complex(c) ; // viene invocato il copy constructor

    Test_reference_complex(c) ; // viene passato solo il reference
```

```
    return EXIT_SUCCESS;
}
```

complex.hpp

- Nome file: complex.hpp
- Contenuto: definizione della strutttura complex

```
#ifndef _COMPLEX_HPP
#define _COMPLEX_HPP

#include <iostream>


using namespace std;


class Complex {

private:
double real ;
double img ;


public:


Complex(void) {
cout << "invoco il costruttore di default" << "\n";
real = 0 ;
img  = 0 ;
} ;

Complex(double real_init, double img_init) {
cout << "invoco il costruttore double double" << "\n";
real = real_init ;
img  = img_init ;
} ;

Complex::Complex( const Complex& c ) {
cout << "invoco il costruttore di copia" << "\n";
real = c.real;
img = c.img;
} ;


~Complex(void) {
```

```
cout << "invoco il distruttore " << "\n";
} ;


inline Complex operator+(const Complex &c) {
     return Complex(this->real + c.real, this->img + c.img);
} ;

inline Complex operator+(const double real_number) {
     return Complex(this->real + real_number, this->img);
} ;


Complex &operator=( const Complex &c) {
cout << "invoco operatore =" << "\n";
real = c.real ;
img = c.img ;
return (*this) ;
} ;


void Print(void) {
cout << "z = [" << real << "," << img << "] \n";
} ;

} ;

inline Complex operator+(double left_side, Complex &c) ;

int Test_copy_complex(Complex input) ;
int Test_reference_complex(Complex &input) ;

#endif
```

complex.cpp

- Nome file: complex.cpp
- Contenuto: implementazione delle funzioni per la manipolazione delle strutture di tipo complex.

```
// ------------------ Begin esempio_16 ---------------
//
```

```
// Utilizzo di una funzione


#include <cstdlib>
#include <iostream>

#include "complex.hpp"


using namespace std;




inline Complex operator+(double left_side, Complex &c) {
    return c + left_side;
}


int Test_copy_complex(Complex input) {
input.Print() ;
return 0 ;
}

int Test_reference_complex(Complex &input) {
input.Print() ;
return 0 ;
}
```

È utile osservare come nel esempio riportato sopra il significato di numero complesso, la sua idea, la sua essenza, sia stata trasferita all'interno della classe, a differenza di quanto accadeva nel caso del C tramite il costrutto struttura, dove il "significato" rimaneva al di fuore del codice, nella mente dell'utente chiamato ad utilizzare quella struttura, con tutti i limiti ed i problemi che questo comportava.

Anche nel caso dei polinomi è possibile portare a termine il progetto di definire un nuovo tipo di variabile che ne consenta la rappresentazione e la gestione. Vedremo che anche in questo caso è fondamentale poter ridefinire gli operatori di base $+$, $-$ ecc.

Un aspetto da sottolineare è come l'utente che voglia utilizzare la classe Polinomio, non debba preoccuparsi delle procedure d'allocazione necessarie per la gestione dell'oggetto. Queste infatti vengono svolte in maniera trasparente dal costruttore dell'oggetto, senza che l'utente ne sia consapevole o se

ne debba curare. Stesse considerazioni valgono per la deallocazione delle strutture soggiacenti l'oggetto polinomio. Una notevole differenza rispetto alla programmazione procedurale e all'utilizzo delle strutture, dove l'utente comunque era coinvolto negli aspetti gesionali (allocazione / inizializzazione /deallocazione) delle strutture. Questo aspetto conferisce alla programmazione ad oggetti una notevole potenza, in quanto permette agli utenti di concentrarsi sul proprio core business, senza doversi preoccupare di aspetti gesionali di basso livello, portando quindi ad una maggiore efficienza ed una maggiore sicurezza.

5.3.6 Esempio di utilizzo del polimorfismo, ovvero programmare riferendosi all'interfaccia e non all'implementazione specifica.

In questo capitolo abbiamo discusso dei fondamenti della programmazione ad oggetti. Questa si poggia su tre pilastri: l'incapsulamento / mascheramento dei dati, l'ereditarietà (di cui si è detto nel capitolo [**?**]) ed il polimorfismo, che sarà l'oggetto del presente capitolo.

Il codice che segue mostra come implementare una classe astratta Vettore, utilizzata poi nel main per calcolare il prodotto scalare tra due vettori. Dalla classe astratta Vettore, vengono derivate due classi relative a due diverse strategie implementative dei vettori:

- i vettori standard basati su un array interno di tipo double (classe Vettore_std).
- i vettori intesi come liste (classe Vettore_lista). In questo caso un singolo vettore è ottenuto come una lista di elementi di oggetti di tipo Elemento_lista. Un oggetto Elemento_lista contiene un intero che rappresenta la posizione ed un valore double che rappresenta il valore associato a quella posizione. La lista si caratterizza per il fatto che ogni elemento della lista conosce l'indirizzo dell'elemento precedente e di quello successivo. Una lista così costruita può essere utilizzata per definire un vettore. Partendo da un elemento della lista è possibile percorrerla nelle due direzioni (dal precedente al successivo o dal successivo al precedente), scorrendo in tal modo gli elementi del vettore. L'utilizzo delle liste per definire i vettori risulta particolarmente vantaggioso nel caso di vettori sparsi (vettori di grandissime dimensioni in cui la maggior parte degli elementi sia valorizzata a zero). In questi casi viene definita una lista contenente solo gli elementi non nulli. Gli elementi a zero del vettore non vengono rappresentati tramite alcun elemento nella lista. In questo modo si ha un notevole risparmio di memoria ed una maggiore efficienza computazionale.

La struttura dei file è organizzata come segue:

- classe Elemento_lista in modalità invasiva (.hpp)
- classe Elemento_lista in modalità non invasiva (.hpp)
- classe Vettore (.hpp e .cpp)
- classe Vettore_std (.hpp e .cpp)
- classe Vettore_lista (.hpp e .cpp)

- main (.cpp) con un esempio di allocazione di due vettori (di tipo standard e di tipo lista) di cui viene calcolato il prodotto interno.

5.3.6.a Programma: classi per la gestione dei vettori (vedi codice libreria 03.05.01)

main.cpp

- Nome file: main.cpp
- Contenuto: main per la manipolazione di alcuni vettori

```
#include <cstdlib>
#include <iostream>

#include "vettore_std.hpp"
#include "vettore_lista.hpp"


using namespace std;
extern int run ;
int run = 0 ;
int Esempio_calcolo_tra_vettori(Vettore &v_1, Vettore &v_2) ;


int main(void) {

   int dim ;

   // Legge il valore e lo copia in n come scanf()
   //
   cout << "Inserire dim del vettore \n" ;
   cin >> dim;



   // Creo dei vettori utilizzando gli oggetti standard
   //
   cout << "standard" ;
   Vettore_std v_std_1(dim) ;
   Vettore_std v_std_2(dim);
   Esempio_calcolo_tra_vettori(v_std_1, v_std_2) ;



   // Creo dei vettori utilizzando gli oggetti lista
   //
```

```
   cout << "lista" ;
   Vettore_lista v_lst_1(dim) ;
   Vettore_lista v_lst_2(dim);
   Esempio_calcolo_tra_vettori(v_lst_1, v_lst_2) ;


}


int Esempio_calcolo_tra_vettori(Vettore &v_1, Vettore &v_2) {


   // Modifico dati dei vettori
   //
   for(long int i=0; i<v_1.Get_dim(); i++) {
      v_1.Set_element(i, 0.0 + i*0.000) ;
      v_2.Set_element(i, 0.0 + i*0.000) ;
   }


   v_1.Set_element(v_1.Get_dim()/2, 2.0) ;
   v_2.Set_element(v_2.Get_dim()/2, 4.0) ;

   // Stampa dei vettori
   //
   // v_1.Stampa() ;
   // v_2.Stampa() ;

   // Prodotto interno dei vettori
   //
   for(int k=0; k<100; k++) {
       v_1 * v_2 ;
   }

   cout << "   prodotto interno = " << v_1 * v_2 << " \n" ;

}
```

vettore.hpp

- Nome file: vettore.hpp
- Contenuto: definizione della classe vettore astratta

```
//
// Dichiarazione della classe Vettore
```

```
#ifndef _VETTORE_HPP
#define _VETTORE_HPP


# define DEFAULT_DIM 1


class Vettore {

   private:
      long int dim ;
      int Set_dim(long int _dim) ;

   protected:

   public:

   // Costruttori
   //
   Vettore(void) {  // in line
      Set_dim(DEFAULT_DIM) ;
   };

   Vettore(long int _dim) {
      Set_dim(_dim) ;
   } ;

   Vettore(const Vettore &obj) {
      Set_dim(obj.dim) ;
   } ;

   // Distruttore
   //
   virtual ~Vettore() ;

   // Funzioni virtuali di input/output
   //
   virtual int Set_element(long int posizione, double valore) = 0 ;

   virtual double Get_element(long int posizione) = 0 ;

   // funzioni pubbliche varie
   //
   long int Get_dim(void) {
      return dim ;
   } ;

   int Stampa(void) ;
```

```
   int Reset_vettore_a_zero(void) ;

   // ridefinizione operatori
   //
   virtual double operator*(Vettore &obj) ;

} ;

#endif
```

vettore.cpp

- Nome file: vettore.cpp
- Contenuto: implementazione funzioni della classe Vettore

```
// Implementazione metodi della classe Vettore


#include <cstdlib>
#include <iostream>

#include "vettore.hpp"


using namespace std;


// ----------> Class:           Vettore
// ----------> Function name:   ~Vettore
// ----------> Description:     Distruttore di Vettore
//
Vettore::~Vettore(void) {
   cout << "Distruttore di Vettore \n" ;
}


// ----------> Class:           Vettore
// ----------> Function name:   Set_dim
// ----------> Description:     Set dim del vettore
//
int Vettore::Set_dim(long int _dim) {

   if(_dim > 0) {
      dim = _dim ;
      return 0 ;
   }
```

```
   else {
      cout << "Errore _dim = " << _dim << "non ammesso \n" ;
      dim = DEFAULT_DIM ;
      return 1 ;
   }

}


// ----------> Class:           Vettore
// ----------> Function name:   Stampa
// ----------> Description:     Stampa del vettore
//
int Vettore::Stampa(void) {

   cout << "Stampa del vettore \n" ;

   for(long int i=0; i<Get_dim(); i++) {
      cout << "      Vettore[" << i << "] = " << Get_element(i) << " \n" ;
   }

}

// ----------> Class:           Vettore
// ----------> Function name:   Reset_vettore_a_zero
// ----------> Description:     Resetta tutti gli
//                              elementi del vettore a zero.
//
int Vettore::Reset_vettore_a_zero(void) {

   for(int i=0; i<Get_dim(); i++) {
      Set_element(i, 0) ;
   }
}


// ----------> Class:           Vettore
// ----------> Function name:   operator*
// ----------> Description:     Prodotto interno
//
double Vettore::operator*(Vettore &obj) {

   double result = 0 ;

   for(int i=0; i<Get_dim(); i++) {
      result = result + Get_element(i) * obj.Get_element(i)  ;
   }

   return result ;
```

```
}
```

vettore_std.hpp

- Nome file: vettore_std.hpp
- Contenuto: definizione della classe Vettore_std (derivata da Vettore)

```
//
// Dichiarazione della classe Vettore_std


#ifndef _VETTORE_STD_HPP
#define _VETTORE_STD_HPP

#include "vettore.hpp"


class Vettore_std : public Vettore {

   private:
      // dati privati
      //
      double *pointer_vettore ;

      // funzioni private
      //
      int Alloc_vettore_std() ;

   protected:

   public:

   // Costruttori
   //
   Vettore_std(void) : Vettore() {  // in line
      Alloc_vettore_std() ;
      Reset_vettore_a_zero() ;
   };

   Vettore_std(long int _dim) : Vettore(_dim) {  // in line
      Alloc_vettore_std() ;
      Reset_vettore_a_zero() ;
   };

   Vettore_std(long int _dim, double *_pointer_vettore) ;
```

```
   Vettore_std(const Vettore_std &obj) ;


   // Distruttore
   //
   virtual ~Vettore_std() ;

   // Funzioni virtuali di input/output
   //
   virtual int Set_element(long int posizione, double valore) ;

   virtual double Get_element(long int posizione) ;

} ;

#endif
```

vettore_std.cpp

- Nome file: vettore_std.cpp
- Contenuto: implementazione funzioni della classe Vettore_std

```
//
// Implementazione metodi della classe Vettore_std


#include <cstdlib>
#include <iostream>

#include "vettore_std.hpp"


using namespace std;


// ----------> Class:           Vettore_std
// ----------> Function name:   Vettore_std
// ----------> Description:     Costruttore di Vettore_std
//
Vettore_std::Vettore_std(long int _dim, double *_pointer_vettore) : Vettore(_dim) {

   Alloc_vettore_std() ;

   for(long int i=0; i< Get_dim(); i++) {
      Set_element(i, _pointer_vettore[i]) ;
   }
```

```
}

// ----------> Class:           Vettore_std
// ----------> Function name:   Vettore_std
// ----------> Description:     Costruttore di copia per la
//                              classe Vettore_std
//
Vettore_std::Vettore_std(const Vettore_std &obj) : Vettore( (Vettore &) obj) {

   Alloc_vettore_std() ;

   for(long int i=0; i< Get_dim(); i++) {
      Set_element(i, obj.pointer_vettore[i]) ;
   }
}


// ----------> Class:           Vettore_std
// ----------> Function name:   ~Vettore_std
// ----------> Description:     Distruttore di Vettore_std
//
Vettore_std::~Vettore_std(void) {
   cout << "Distruttore di Vettore_std \n" ;
   delete pointer_vettore ;
}


// ----------> Class:           Vettore_std
// ----------> Function name:   Alloc_init_vettore_std
// ----------> Description:     Alloca la memoria per
//                              il vettore
//
int Vettore_std::Alloc_vettore_std(void) {

   if(Get_dim() > 0) {
      // Alloco vettore
      //
      pointer_vettore = new double[Get_dim()] ;

      // Fine metodo
      //
      return 0 ;
   }
   else {
      // Inizializzo il pointer a NULL
      //
      pointer_vettore = NULL ;

      // Fine metodo
```

```
      //
      return 1 ;
   }
}


// ----------> Class:           Vettore_std
// ----------> Function name:   Set_element
// ----------> Description:     Set di un elemento del vettore_std
//
int Vettore_std::Set_element(long int numero_elemento, double valore) {

   if(numero_elemento >= 0 && numero_elemento <Get_dim()) {
      pointer_vettore[numero_elemento] = valore ;
      return 0 ;
   }
   else {
      cout << "Error in Vettore_std::Set_elemento, numero_elemento = "
           << numero_elemento << " e' fuori dal range \n" ;
      return 1 ;
   }
}

// ----------> Class:           Vettore_std
// ----------> Function name:   Get_element
// ----------> Description:     Get di un elemento del vettore_std
//
double Vettore_std::Get_element(long int numero_elemento) {

   if(numero_elemento >= 0 && numero_elemento <Get_dim()) {
      return pointer_vettore[numero_elemento] ;
   }
   else {
      cout << "Error in Vettore_std::Get_elemento, numero_elemento = "
           << numero_elemento << " e' fuori dal range \n" ;
      return 0.0 ;
   }
}
```

elemento_lista_invasiva.hpp

- Nome file: elemento_lista_invasiva.hpp
- Contenuto: definizione della classe Elemento_lista_invasiva (classe utilizzata da Vettore_lista)

```
// ------------------ Begin esempio_lista_invasiva ---------------
//
```

```
// Dichiarazione di una classe


#ifndef _ELEMENTO_LISTA_INVASIVA_HPP
#define _ELEMENTO_LISTA_INVASIVA_HPP

#include <iostream>


using namespace std;

// Esempio di lista invasiva doppiamente "linkata"
//
class Elemento_lista {

   private:

      // dati privati
      //
      double valore ;
      long int posizione ;

      Elemento_lista *next;
      Elemento_lista *previous;


   protected:

   public:

   // Costruttore
   //
   Elemento_lista(void) {  // in line
      previous = NULL ;
      next = NULL ;
      Set_posizione(0) ;
      Set_valore(0) ;
   };

   Elemento_lista(long int _posizione, double _valore) {
      previous = NULL ;
      next = NULL ;
      Set_posizione(_posizione) ;
      Set_valore(_valore) ;
   } ;


   // Distruttore
   //
```

```
virtual ~Elemento_lista() {
   // cout << "Elimino elemento: " << posizione << "\n" ;
} ;


// Funzioni di interrogazione dell'oggetto
//

double Get_valore(void) {
   return valore ;
} ;

long int Get_posizione(void) {
   return posizione ;
} ;


Elemento_lista *Get_next(void) {
   return next ;
} ;


Elemento_lista *Get_previous(void) {
   return previous ;
};


// Funzioni di setting dell'oggetto
//
int Set_valore(double _valore) {
   valore = _valore ;
   return 0 ;
} ;

int Set_posizione(long int _posizione) {
   if(_posizione >= 0) {
      posizione = _posizione ;
      return 0 ;
   }
   else {
      return 1 ;
   }
} ;


int Set_next(Elemento_lista *_next) {
   next = _next ;

   if(next != NULL) {
```

```
         next->previous = this ;
      }
   } ;


   int Set_previous(Elemento_lista *_previous) {
      previous = _previous ;

      if(previous != NULL) {
         previous->next = this ;
      }
   };


} ;

#endif


// ------------------ End esempio_lista_invasiva -----------------
```

elemento_lista_non_invasiva.hpp

- Nome file: elemento_lista_non_invasiva.hpp
- Contenuto: definizione della classe Elemento_lista_non_invasiva (classe utilizzata da Vettore_lista)

```
// ------------------ Begin esempio_lista_non_invasiva ---------------
//
// Dichiarazione di una classe


#ifndef _ELEMENTO_LISTA_NON_INVASIVA_HPP
#define _ELEMENTO_LISTA_NON_INVASIVA_HPP

class Contenitore {

   private:

      // dati privati
      //
      double valore ;
      long int posizione ;

   protected:

   public:
```

```
      // Costruttore
      //
      Contenitore(void) {  // in line
         valore = 0 ;
         posizione = 0 ;
      };

      // Distruttore
      //
      virtual ~Contenitore() {
      } ;


      // Funzioni di interrogazione dell'oggetto
      //

      double Get_valore(void) {
         return valore ;
      } ;

      long int Get_posizione(void) {
         return posizione ;
      } ;




      // Funzioni di setting dell'oggetto
      //
      int Set_valore(double _valore) {
         valore = _valore ;
         return 0 ;
      } ;

      int Set_posizione(long int _posizione) {
         posizione = _posizione ;
         return 0 ;
      } ;

} ;




// Esempio di lista invasiva doppiamente "linkata"
//
class Elemento_lista {
```

```
private:

   // dati privati
   //
   Contenitore *dati ;

   Elemento_lista *next;
   Elemento_lista *previous;


protected:

public:

   // Costruttore
   //
   Elemento_lista(void) {  // in line
      dati = new Contenitore() ;
      previous = NULL ;
      next = NULL ;
   };

   Elemento_lista(long int _posizione, double _valore) {
      dati = new Contenitore() ;
      previous = NULL ;
      next = NULL ;
      Set_posizione(_posizione) ;
      Set_valore(_valore) ;
   } ;


   // Distruttore
   //
   virtual ~Elemento_lista() {
      delete dati ;
   } ;


   // Funzioni di interrogazione dell'oggetto
   //

   double Get_valore(void) {
      return dati->Get_valore() ;
   } ;

   long int Get_posizione(void) {
      return dati->Get_posizione() ;
   } ;
```

```
      Elemento_lista *Get_next(void) {
         return next ;
      } ;

      Elemento_lista *Get_previous(void) {
         return previous ;
      };


      // Funzioni di setting dell'oggetto
      //
      int Set_valore(double _valore) {
         if(dati != NULL) {
            dati->Set_valore(_valore) ;
            return 0 ;
         }
         else {
            return 1 ;
         }
      } ;

      int Set_posizione(long int _posizione) {
         if(dati != NULL && _posizione >= 0) {
            dati->Set_posizione(_posizione) ;
            return 0 ;
         }
         else {
            return 1 ;
        }
      } ;

      int Set_next(Elemento_lista *_next) {
         next = _next ;
         if(next != NULL) {
            next->previous = this ;
         }
      } ;


      int Set_previous(Elemento_lista *_previous) {
         previous = _previous ;
         if(previous != NULL) {
            previous->next = this ;
         }
      };

} ;
```

```
#endif


// ------------------ End esempio_lista_non_invasiva -----------------
```

vettore_lista.hpp

- Nome file: vettore_lista.hpp
- Contenuto: definizione della classe Vettore_lista (derivata da Vettore)

```
//
// Dichiarazione della classe Vettore_lista


#ifndef _VETTORE_LISTA_HPP
#define _VETTORE_LISTA_HPP


#define CURSORE_CENTRATO_SU_POSIZIONE    0
#define CURSORE_A_SINISTRA_DI_POSIZIONE  -1
#define CURSORE_A_DESTRA_DI_POSIZIONE    +1
#define CURSORE_INVALIDO                 -100

#include "vettore.hpp"
// #include "elemento_lista_non_invasiva.hpp"
#include "elemento_lista_invasiva.hpp"

class Vettore_lista : public Vettore {

   private:

      // dati privati
      //
      Elemento_lista *elemento_lista_corrente ;
      int counter_elementi ;


      // funzioni private
      //
      int Alloc_vettore_lista() ;


   protected:

   public:

   // Costruttore
```

```
   //
   Vettore_lista(void) : Vettore() {  // in line
      Alloc_vettore_lista() ;
   };

   Vettore_lista(long int _dim) : Vettore(_dim) {
      Alloc_vettore_lista() ;
   } ;

   Vettore_lista(long int _dim, double *_pointer_vettore) ;

   Vettore_lista(Vettore_lista &obj) ;

   // Distruttore
   //
   virtual ~Vettore_lista() ;


   // Funzioni virtuali di input/output
   //
   virtual int Set_element(long int posizione, double valore) ;

   virtual double Get_element(long int posizione) ;

   // ridefinizione operatori
   //
   virtual double operator*(Vettore &obj) ;


   // Funzioni specifiche di Vettore_lista
   //
   int Posiziona_cursore(long int posizione) ;

   Elemento_lista *Get_cursore(void) {
      return elemento_lista_corrente ;
   } ;

   int Set_cursore(Elemento_lista *_elemento_lista_corrente) ;

   void Set_cursore_ultimo_elemento_valido(void) ;

   void Set_cursore_primo_elemento_valido(void) ;

   int Inserisci_elemento(long int posizione) ;

} ;

#endif
```

vettore_lista.cpp

- Nome file: vettore_lista.cpp
- Contenuto: implementazione funzioni della classe Vettore_lista

```
//
// Implementazione metodi della classe Vettore_lista


#include <cstdlib>
#include <iostream>

#include "vettore_lista.hpp"
extern int run  ;

using namespace std;


// ----------> Class:           Vettore_lista
// ----------> Function name:   Vettore_lista
// ----------> Description:     Costruttore di Vettore_lista
//
Vettore_lista::Vettore_lista(long int _dim, double *_pointer_vettore) : Vettore(_dim) {

   Alloc_vettore_lista() ;

   for(int i=0; i< Get_dim(); i++) {
      Set_element(i, _pointer_vettore[i]) ; ;
   }
}

// ----------> Class:           Vettore_lista
// ----------> Function name:   Vettore_lista
// ----------> Description:     Costruttore di copia per la
//                              classe Vettore_lista
//
Vettore_lista::Vettore_lista(Vettore_lista &obj) : Vettore( (Vettore &) obj) {

   Alloc_vettore_lista() ;

   for(long int i=0; i< Get_dim(); i++) {
      Set_element(i, obj.Get_element(i)) ;
   }
}


// ----------> Class:           Vettore_lista
// ----------> Function name:   ~Vettore_lista
```

```
// ----------> Description:     Distruttore di Vettore_lista
//
Vettore_lista::~Vettore_lista(void) {

   int status_cursore = 0 ;

   cout << "Distruttore di Vettore_lista \n" ;

   Elemento_lista *elemento_lista_next = NULL ;

   Set_cursore_primo_elemento_valido() ;

   status_cursore = 0 ;

   while(status_cursore == 0) {
      elemento_lista_next = Get_cursore()->Get_next() ;
      delete Get_cursore() ;
      counter_elementi-- ;
      status_cursore = Set_cursore(elemento_lista_next) ;
   } ;
}



// ----------> Class:           Vettore_lista
// ----------> Function name:   Set_element
// ----------> Description:     Set di un elemento del vettore_lista
//
int Vettore_lista::Set_element(long int posizione, double valore) {

   if(posizione >=0 && posizione < Get_dim()) {
      if(Posiziona_cursore(posizione) == CURSORE_CENTRATO_SU_POSIZIONE) {
         // esiste un elemento della lista nella posizione
         // indicata
         //
         Get_cursore()->Set_valore(valore) ;
      }
      else {
         // non esiste un elemento della lista nella posizione
         // indicata. Lo creo solo se il valore e' non nullo!
         if(valore == 0) {
         }
         else {
            Inserisci_elemento(posizione) ;
            Set_element(posizione, valore) ;
         }
      }
   }
   else {
```

```
      cout << "Error in Vettore_lista::Set_elemento, posizione = "
           << posizione << " e' fuori dal range \n" ;
      return 1 ;
   }
}



// ----------> Class:           Vettore_lista
// ----------> Function name:   Get_element
// ----------> Description:     Get di un elemento del vettore_lista
//
double Vettore_lista::Get_element(long int posizione) {

   if(posizione >=0 && posizione <Get_dim()) {
      if(Posiziona_cursore(posizione) == CURSORE_CENTRATO_SU_POSIZIONE) {
         return Get_cursore()->Get_valore() ;
      }
      else {
         return 0.0 ;
      }
   }
   else {
      cout << "Error in Vettore_lista::Get_elemento, posizione = "
           << posizione << " e' fuori dal range \n" ;
      return 0.0 ;
   }
}



// ----------> Class:           Vettore_lista
// ----------> Function name:   Inserisci_elemento
// ----------> Description:
//
int Vettore_lista::Inserisci_elemento(long int posizione) {

   Elemento_lista *nuovo_elemento_lista = NULL ;
   Elemento_lista *elemento_lista_next = NULL ;
   Elemento_lista *elemento_lista_previous = NULL ;

   int status_cursore = 0 ;

   status_cursore = Posiziona_cursore(posizione) ;

   if(status_cursore == CURSORE_CENTRATO_SU_POSIZIONE) {
      // elemento gia' esistente nella posizione specificata
      //
      return -1 ;
   }
```

```
   if(status_cursore == CURSORE_INVALIDO) {
      // posizione specificata invalida
      //
      return -2 ;
   }

   elemento_lista_next = Get_cursore()->Get_next() ;
   elemento_lista_previous = Get_cursore()->Get_previous() ;

   // Creo nuovo elemento
   //
   nuovo_elemento_lista = new Elemento_lista(posizione, 0) ;

   if(status_cursore == CURSORE_A_SINISTRA_DI_POSIZIONE) {

      // Collego il nuovo elemento nella lista
      //
      nuovo_elemento_lista->Set_previous(Get_cursore()) ;
      nuovo_elemento_lista->Set_next(elemento_lista_next) ;

   }
   else if(status_cursore == CURSORE_A_DESTRA_DI_POSIZIONE) {

      // Collego il nuovo elemento nella lista
      //
      nuovo_elemento_lista->Set_previous(elemento_lista_previous) ;
      nuovo_elemento_lista->Set_next(Get_cursore()) ;

   }

   // incremento il contatore di elementi della lista
   //
   counter_elementi ++ ;

   // Posiziono il cursore sul nuovo elemento
   //
   Set_cursore(nuovo_elemento_lista) ;

   return 0 ;
}



// ----------> Class:           Vettore_lista
// ----------> Function name:   Posiziona_cursore
// ----------> Description:     Posiziona il cursore
//
int Vettore_lista::Posiziona_cursore(long int posizione) {
```

```
   if(posizione >= 0 && posizione < Get_dim()) {

      if( posizione == Get_cursore()->Get_posizione() ) {
         return CURSORE_CENTRATO_SU_POSIZIONE ;
      }
      else if(posizione > Get_cursore()->Get_posizione()) {

         while(Get_cursore()->Get_next() != NULL &&
               posizione >= Get_cursore()->Get_next()->Get_posizione()) {
            // Movimento indietro
            //
            Set_cursore(Get_cursore()->Get_next()) ;
         }

         if(posizione == Get_cursore()->Get_posizione()) {
            return CURSORE_CENTRATO_SU_POSIZIONE ;
         }
         else {
            return CURSORE_A_SINISTRA_DI_POSIZIONE ;
         }
      }
      else if(posizione < Get_cursore()->Get_posizione()) {
         while(Get_cursore()->Get_previous() != NULL &&
               posizione <= Get_cursore()->Get_previous()->Get_posizione()) {
            // Movimento indietro
            //
            Set_cursore(Get_cursore()->Get_previous()) ;
         }
         if(posizione == Get_cursore()->Get_posizione()) {
            return CURSORE_CENTRATO_SU_POSIZIONE ;
         }
         else {
            return CURSORE_A_DESTRA_DI_POSIZIONE ;
         }
      }
      else {
         cout << "Errore inatteso \n" ;
         return CURSORE_INVALIDO ;
      }
   }
   else {
      cout << "Index out of range \n" ;
      return CURSORE_INVALIDO ;
   }
}


// ----------> Class:           Vettore_lista
```

```
// ----------> Function name:   Set_cursore_ultimo_elemento
// ----------> Description:
//
void Vettore_lista::Set_cursore_ultimo_elemento_valido(void) {

   while(Get_cursore()->Get_next() != NULL) {
      Set_cursore(Get_cursore()->Get_next()) ;
   }
}

// ----------> Class:           Vettore_lista
// ----------> Function name:   Set_cursore_primo_elemento
// ----------> Description:
//
void Vettore_lista::Set_cursore_primo_elemento_valido(void) {

   while(Get_cursore()->Get_previous() != NULL) {
      Set_cursore(Get_cursore()->Get_previous()) ;
   }
}


// ----------> Class:           Vettore_lista
// ----------> Function name:   Alloca_vettore_lista
// ----------> Description:
//
int Vettore_lista::Alloc_vettore_lista(void) {


   Elemento_lista *nuovo_elemento_lista = NULL ;

   counter_elementi = 0 ;

   nuovo_elemento_lista = new Elemento_lista(0, 0.0) ;

   nuovo_elemento_lista->Set_previous(NULL) ;
   nuovo_elemento_lista->Set_next(NULL) ;

   counter_elementi ++ ;

   Set_cursore(nuovo_elemento_lista) ;

   return 0 ;

}


// ----------> Class:           Vettore_lista
// ----------> Function name:   Set_cursore
```

```
// ----------> Description:
//
int Vettore_lista::Set_cursore(Elemento_lista *_elemento_lista_corrente) {

   if(_elemento_lista_corrente != NULL) {
      if(_elemento_lista_corrente->Get_posizione() >= 0 &&
         _elemento_lista_corrente->Get_posizione() < Get_dim()) {

            elemento_lista_corrente = _elemento_lista_corrente ;
            return 0 ;
      }
      else {
            return 1 ;
      }
   }
   else {
      return 1 ;
   }
}

// ----------> Class:           Vettore_lista
// ----------> Function name:   operator*
// ----------> Description:     Prodotto interno
//
double Vettore_lista::operator*(Vettore &obj) {

   double result = 0 ;
   long int posizione = 0 ;
   int status_cursore = 0 ;

   // cout << "prodotto ottimizzato (" << counter_elementi << ")\n " ;

   Set_cursore_primo_elemento_valido() ;

   status_cursore = 0 ;

   while(status_cursore == 0) {
      posizione = Get_cursore()->Get_posizione() ;
      result = result + Get_cursore()->Get_valore() * obj.Get_element(posizione) ;
      status_cursore = Set_cursore(Get_cursore()->Get_next()) ;
   } ;

   return result ;
}
```

5.3.6.b Commenti

Di seguito alcuni commenti:

(1) Si noti come la funzione che effettua il prodotto interno tra i due vettori d'esempio, possa utilizzare indifferentemente oggetti di tipo Vettore_std o di tipo Vettore_lista. In altri termini il codice presente nel main, per gestire il prodotto interno tra vettori, risulta totalmente astratto, non dipendente dall'implementazione degli oggetti. Questo perché nella scrittura del codice ci si è riferiti all'interfaccia della classe astratta Vettore e non all'implementazione specifica utilizzata nelle due classi concrete Vettore_std e Vettore_lista. Questo è un tipico esempio di uso avanzato del polimorfismo.
Il polimorfismo consiste nel fatto che solo in fase di run-time verrano chiamate (ovvero risolte ed eseguite) le funzioni appropriate in base al tipo di oggetto reale coinvolto. In altre parole nel codice che effettua il prodotto scalare, vengono utilizzate delle funzioni membro di Vettore, in modo totalmente astratto e solo durante il run-time verranno risolte in maniera adeguata in base agli oggetti effettivamente passati alla funzione.
Il polimorfismo costituisce il terzo pilastro della programmazione ad oggetti ed è l'architrave fondamentale per garantire una piena ed efficace riutilizzabilità del codice. (Nell'esempio scritto sopra, il codice per la gestione del prodotto interno nel main, non va modificato se ad esempio si dovesse decidere di inserire una terza classe derivata da Vettore, con un'implementazione differente dalle prime due. In tal modo è possibile estendere il codice, inserendo nuove implementazioni senza dover modificare in alcun punto il codice già scritto.)

(2) Si osservi l'uso delle funzioni virtual nel codice. Queste sono essenziali per garantire l'utilizzo pieno del polimorfismo.

(3) Entrando nel merito dell'esempio, va osservato come l'utilizzo di Vettore_lista risulti nettamente preferibile a quello di Vettore_std nel caso di vettori sparsi (vettori di grandi dimensioni in cui la maggior parte degli elementi sia nulla). Questo sia dal punto di vista della memoria allocata, sia in termini di performance della cpu.

(4) Si noti la ridefinizione dell'operatore prodotto nella classe Vettore al fine di trattare il prodotto interno tra vettori. L'operatore prodotto è stato dichiarato virtual al fine di consentirne la ridefinizione (nell'ottica del polimorfismo) nelle classi a valle.
In effetti nella classe Vettore_lista si è ridefinito l'operatore prodotto in modo da migliorarne le performance. Questo è dunque un esempio di come in alcuni casi sia utile ridefinire un membro di una classe a valle, rispetto a quello omonimo definito nella classe padre, per ragioni riconducibili esclusivamente a questioni di performance di calcolo. Se ad es. si evita di ridefinire il prodotto nella classe Vettore_lista sarà ovviamente possibile calcolare il prodotto interno di due vettori di

tipo lista, ma in tal caso l'algoritmo utilizzato (quello contenuto nella generica classe Vettore) sarà meno efficiente.

5.3.7 Esempio di utilizzo avanzato del polimorfismo.

Il codice scritto sopra comprende come abbiamo visto consente di eseguire il prodotto interno di vettori. Questi possono essere di due tipi fondamentali: vettori standard (stile C ma incapsulati in una classe) e vettori di tipo lista (adatti a trattare vettori sparsi).
Supponiamo che successivamente nasca la necessità di utilizzare un tipo differente di vettore, ad es. si voglia utilizzare la classe vector disponibile nella standard library del C++. In questo caso vi è un problema fondamentale all'utilizzo un oggetto di tipo vector ¡double¿ all'interno della funzione prodotto interno, ovvero il fatto che vector ¡double¿ non è figlio della classe astratta Vettore. In sostanza in tale contesto non è possibile utilizzare il polimorfismo in quanto la classe vector ¡double¿ non ha la stessa interfaccia di Vettore. Questo problema può però essere superato attraverso una tecnica di programmazione ad oggetti nota come Adapter. Tale tecnica costituisce un esempio di design pattern ed è illustrata nell'esempio riportato di seguito.

5.3.7.a Programma: classi per la gestione dei vettori (vedi codice libreria 03.05.02)

main.cpp

- Nome file: main.cpp
- Contenuto: main per la manipolazione di alcuni vettori

```
#include <cstdlib>
#include <iostream>

#include "vettore_std.hpp"
#include "vettore_lista.hpp"
#include "prodotto_interno.hpp"


using namespace std;

int main(void) {

   int dim ;

   // Legge il valore e lo copia in n come scanf()
   //
   cout << "Inserire dim del vettore \n" ;
   cin >> dim;
```

```
   // Creo dei vettori utilizzando i vari costruttori
   //

   Vettore_std v1(dim, 10.0) ;

   v1.Set_elemento(2, 5.0) ;

   Vettore_con_adapter v1(dim, 10.0) ;

   v1.Stampa() ;
   v2.Stampa() ;

   double result ;
   Prodotto_interno_tra_vettori(&result, &v2, &v4) ;

   cout << "prodotto interno = " << result << " \n" ;


}
```

vector_con_adapter.hpp

- Nome file: vector_con_adapter.hpp
- Contenuto: definizione della classe Vector_con_adapter

```
//
// Dichiarazione di una classe


#ifndef _VECTOR_CON_ADAPTER_HPP
#define _VECTOR_CON_ADAPTER_HPP

#include "vettore.hpp"


class Vector_con_adapter : public Vector {

    private:

        // dati privati
        //
        vector <double> *vector_std_library ;
```

```
protected:

public:

    // Costruttore
    //
    Vector_con_adapter(void) : Vector() {  // in line
        vector_std_library = new vector <double> ;
    };

    Vector_con_adapter(int _dim) : Vector() {
        if(_dim > 0) {
            vector_std_library = new vector <double> (_dim)  ;
        }
        else {
            vector_std_library = NULL ;
        }
    } ;

    Vector_con_adapter(int _dim, double *_pointer_vettore) : Vector()  {

        vector_std_library = new vector <double> ;

        for( int i = 0; i < _dim; ++i ) {
            vector_std_library->push_back( _pointer_vettore[i] );
        }
    } ;

    Vector_con_adapter(const Vector_con_adapter &obj) : Vector( (Vector &) obj) {
            vector_std_library = new vector <double>(obj.vector_std_library->size()) ;

        for( int i = 0; i < obj.vector_std_library->size(); ++i ) {
            vector_std_library->push_back( obj.vector_std_library->at(i) );
        }
    } ;


    // Distruttore
    //
    virtual ~Vector_con_adapter() {
        delete vector_std_library ;
    } ;


    // Funzioni di input/output
    //
    virtual int Set_elemento(int posizione, double valore) {
        if(posizione>=0 && posizione < vector_std_library->size()) {
            vector_std_library->at(posizione) = valore ;
```

```
            }
        } ;

        virtual double Get_elemento(int posizione) {
            if(posizione>=0 && posizione < vector_std_library->size()) {
                return vector_std_library->at(posizione) ;
            }
        } ;


} ;

#endif
```

(1) Si noti come la nuova classe Vector_con_adapter non faccia altro che avvolgere un oggetto di tipo vector ⟨double⟩, andando a rivestirne l'interfaccia. La classe Vector_con_adapter essendo derivata da Vector ne eredità l'interfaccia e d'altra parte possedendo al proprio interno un pointer a vector ⟨double⟩ consente di gestire un vettore utilizzando la standard library del C++.
Il programmatore della classe dovrà solo preoccuparsi di implementare le funzioni di interfacci per "connettere" l'interfaccia della classe con l'interndo dell'oggetto costituita dal pointer di tipo vector ⟨double⟩.
Questa metodologia di programmazione è nota come adapter in quanto quello realizzato è un vero e proprio adattatore di classe, ovvero consente di rivestire o riconvertire l'interfaccia di una classe (nel nostro esempio quella di vector ⟨double⟩) in quella di un'altra (nel nostro esempio Vector).

(2) Una volta adattata l'interfaccia di vector ⟨double⟩ a quella di Vector, ecco che saremo in grado di utilizzare gli oggetti di tipo Vector_con_adapter nella funzione per il calcolo del prodotto interno.

(3) Si osservi infine come l'aggiunta di una terza tipologia di vettori (derivati dalla classe astratta Vettore), sia avvenuta in maniera assolutamente modulare, senza richiedere la riscrittura del codice pre-esistente (nel nostro caso la funzione per il calolo del prodotto interno).

In conclusione in questo capitolo abbiamo mostrato come il polimorfismo consenta di aggiungere in un secondo momento alla libreria una nuova sottotipologia di oggetti, senza per questo modificare il codice pre-esistente. Nel caso in cui la tipologia di oggetto da inserire è stata sviluppata da una parte terza e perciò è caratterizzata da un interfaccia non compatibile da quella richiesta, è sempre possibile ricorrere ad un adattatore (adapter).

L'adapter mostrato è un esempio di design pattern, di cui si parlerà più diffusamente nel capitolo successivo.

5.3.8 Design patterns

5.3.8.a Design patterns: il singleton

Un esempio molto famoso e molto semplice di design pattern è il singleton.

5.3.8.b Esempio di definizione di un singleton (vedi codice libreria 04.01)

main.cpp

- Nome file: main.cpp
- Contenuto: main per l'utilizzo del singleton

```
// initialize pointer
//
Singleton_coda_messaggi *Singleton_coda_messaggi::pointer_istanza = NULL ;

int main() {

    Singleton_coda_messaggi *coda_msg ;

    coda_msg = Singleton_coda_messaggi::Get_instance() ;

    coda_msg->Put_msg("messaggio 1") ;
    coda_msg->Put_msg("messaggio 2") ;
    coda_msg->Put_msg("messaggio 3") ;

    coda_msg->Print_all_msg() ;

    return 0;

}
```

singleton.hpp

- Nome file: singleton.hpp
- Contenuto: definizione della classe Singleton

```
//
// Dichiarazione di una classe
```

```
#ifndef _SINGLETON_HPP
#define _SINGLETON_HPP

#include <iostream>


class Singleton_coda_messaggi {

    private:
        static singleton* pointer_istanza ;

        Singleton_coda_messaggi() { };

    public:

        ~Singleton_coda_messaggi() { };

        static Singleton_coda_messaggi* Get_instance() {
            if (pointer_istanza == NULL) {
                pointer_istanza = new Singleton_coda_messaggi() ;
            }

            return pointer_istanza;
        } ;

        int Put_msg(String *_msg) ;

        String Get_msg(int counter) ;
};


#endif
```

5.3.8.c Altri design patterns

Vi sono vari tipi di design patterns. Un valido libro d'approfondimento su questo tema è dato da [27].
Esempi di design pattern (relativamente semplici) sono il decorator e l'observer.

Bibliografia

[1] J. C. Hull, *Opzioni, Futures e altri derivati*, III edizione, Il Sole 24 Ore Libri. In alternativa: J. C. Hull, *Options, Futures and other derivatives*, V edizione USA, Prentice Hall.

[2] P. Wilmott, S. Howison and J. Dewynne, *The Mathematics of Financial Derivatives : A Student Introduction*, Cambridge University Press (1995).

[3] P. Wilmott, *Paul Wilmott on Quantitative Finance*, 2 volumi, John Wiley and Sons Ltd (2000).

[4] R. N. Mantegna and H. E. Stanley, *An Introduction to Econophysics: Correlations and Complexity in Finance*, Cambridge University Press (1999).

[5] J. M. Pimbley, *Physicists in Finance*, Physics Today, **Jan.**, 42 (1997).

[6] L. Bachelier, *Théorie de la spéculation* (Ph.D. thesis in mathematics), Annales Scientifiques de l'Ecole Normale Supérieure **III-17**, 21–86 (1900).

[7] B. B. Mandelbrot, *The Variation of Certain Speculative Prices*, Journal of Business, **36**, 394–419 (1963).

[8] B. B. Mandelbrot, *Fractals and Scaling in Finance*, Springer–Verlag, New York, (1997).

[9] V. Plerou, P. Gopikrishnan, B. Rosenow, L. A. N. Amaral and H. E. Stanley, *Econophysics: Financial Time Series from a Statistical Physics Point of View*, Physica A, **279**, 443–456 (2000).

[10] R. N. Mantegna and H. E. Stanley, *Scaling Behavior in the Dynamics of an Economic Index*, Nature, **376**, 46–49 (1995).

[11] R. N. Mantegna and H. E. Stanley, *Turbulence and Financial Markets*, Nature, **383**. 587–588 (1996).

[12] R. N. Mantegna and H. E. Stanley, *Stock Market Dynamics and Turbulence: Parallel Analysis of Fluctuation Phenomena*, Physica A, **239**, 255–266 (1997).

[13] F. Black e M. Scholes, *The Pricing of Options and Corporate Liabilities*, Journal of Political Economy, **81**, 637–654 (1973).

[14] Molti articoli sono stati pubblicati su questo tema. Alcuni esempi sono: J. Cox and S. Ross, *The valuation of options for alternative stochastic processes*, Journal of Financial Economics, **3**, 145–166 (1976); B. M. Bibby and M. Sørensen, *A hyperbolic diffusion model for stock prices*, Finance and Stochastic, **1**, 25–42 (1997).

[15] S. M. Turnbull and L. M. Wakeman, *A quick algorithm for pricing European average options*, Journal of Financial and Quantitative Finance, **26**, 377–389, (1991).

[16] L. Bouaziz, E. Briys, M. Crouhy *The pricing of forward-starting asian options*, Journal of Banking and Finance, **18**, n. 5, 823–839, (1994).

[17] M. Esposito, *Valutazione di opzioni su medie aritmetiche discrete: una soluzione in forma (quasi) chiusa*, Capital Market Notes, 1998.

[18] M. Airoldi, *A moment expansion approach to option pricing*, Quantitative Finance, (2005).

[19] T. W. Lim, *Performance of recursive integration for pricing European-style Asian options*, preprint.

[20] E. D. Weinberger, *How Close in Close Enough? The Limitations of Monte Carlo in Computing VaR*, preprint (1999).

[21] P. Jackel, *Monte Carlo Methods in Finance*, Wiley Finance, (2002).

[22] P. Glasserman, *Monte Carlo Methods in Financial Engineering*, Springer.

[23] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes in C++ - The Art of Scientific Computing*, Cambridge University Press, (2002), (www.numerical-recipes.com).

[24] D. Brigo, F. Marcurio, *Interest Rate Models: Theory and Practice*, Springer.

[25] Guido Buzzi-Ferraris, *Dal Fortran al C++ - Introduzione alla programmazione orientata agli oggetti applicata a problemi numerici*, Addison Wesley.

[26] Bruce Eckel, *Thinking in C++*, Prentice Hall Inc. Pearson Higher Education.

[27] Gamma, Helm, Johnson e Vlissides *Design Patterns*, Pearson Addison Wesley.

[28] Per un corso on-line sulla programmazione ad oggetti, si faccia riferimento a:
http://www.science.unitn.it/ iori/java/INDEXCPP.html

[29] Sulle convenzioni stilistiche da adottare per una buona programmazione ad oggetti si faccia riferimento a:
http://geosoft.no/development/cppstyle.html

[30] Sulla gestione ed utilizzazione degli smart pointer, una descrizione sintetica è data in:
http://ootips.org/yonat/4dev/smart-pointers.html

Indice analitico