# Valutazione numerica di opzioni con *Mathematica*

*di Maurizio Spadaccino* (maurizio.spadaccino@bancaimi.it)

*alla memoria del prof. Franco Caparrelli*

## Introduzione

*Mathematica*, oltre a costituire un potente foglio di calcolo, è un linguaggio di programmazione interattivo e flessibile che può essere efficacemente sfruttato per la soluzione di molteplici problemi, tra i quali la valutazione delle opzioni, strumenti finanziari derivati che, per la loro complessa natura, possono talora risultare di problematica quantificazione. In questo lavoro verrà approfondito innanzitutto il tema del *pricing* di opzioni tradizionali (*plain-vanilla*) sia di tipo europeo che americano, attraverso i tradizionali modelli di Cox, Ross e Rubinstein e quello di Black & Scholes. In seguito, si affronterà il problema della determinazione del prezzo di opzioni dal payoff più complesso *(exotic options)*, in particolare mediante le simulazioni Monte-Carlo, facilmente implementabili tramite questo software.

## Determinazione del prezzo di Opzioni Europee ed Americane con il Modello Binomiale.

In questa sezione analizzeremo la procedura che consente di giungere al prezzo di opzioni *plain-vanilla* europee ed americane con l'approccio di Cox-Ross-Rubinstein. Va precisato fin d'ora come la scelta di rappresentare anche visivamente le fasi di sviluppo del modello di pricing abbia prevalso sullo sfruttamento delle capacità di *Mathematica*, tramite funzioni recursive, di sviluppare funzioni personalizzate in forma più efficiente ed elegante. Tutto questo al fine di una più chiara comprensione dei vari passaggi richiesti dall'implementazione dell'approccio binomiale.

### Titolo che paga un dividendo continuo.

Supponiamo di voler determinare il prezzo di un'opzione call europea *at the money* che scade tra 6 mesi, avendo previsto una volatilità pari al 25% e con un prezzo corrente del titolo sottostante pari a 50. Ipotizziamo inoltre un tasso d'interesse del 5%.
Introduciamo gli input necessari per la costruzione dell'albero binomiale. In primo luogo, occorre definire la variabile vita residua dell'opzione, **`tm`**, espressa in anni:

```
tm = 6 / 12;
```

L'albero sarà costituito da un certo numero di stadi, **`per`**, supponiamo pari a 5:

```
per = 5;
```

La distanza tra un nodo e quello successivo del nostro albero risulterà pertanto uguale a dt:

```
dt = tm / per
```
$$\frac{1}{10}$$

Inseriamo a questo punto i parametri relativi al tasso privo di rischio ed alla volatilità:

```
riskfree = 0.05; vol = 0.25;
```

Il modello binomiale prevede anche la possibilità di considerare un tasso di dividendo, yield, che rappresenta il dividendo prodotto continuamente dal titolo (o indice) sottostante. Quest'ultimo, nel caso in cui l'attività sottostante l'opzione sia costituita da un contratto futures, sarà uguale al tasso risk-free, riskfree, già sopra inserito. Supponiamo che il titolo non paghi dividendi:

```
yield = 0;
```

Il tasso netto risulterà pertanto dalla differenza tra il tasso privo di rischio ed il dividend yield:

```
rf = riskfree - yield
0.05
```

In ciascun nodo dell'albero, il prezzo del sottostante può crescere di un ammontare pari a u, o scendere di un ammontare uguale a d=1/u. Ad esempio, in corrispondenza del primo nodo, dove il prezzo è dato da a nello sviluppo successivo si potrà arrivare al nodo superiore, dove a oppure al nodo inferiore con a. Il caso discreto per un processo Browniano del prezzo del sottostante porta a valori di *u* pari a a.

```
u = Exp[vol * Sqrt[dt]]
1.08227
d = 1 / u
0.923987
```

Determiniamo quindi le pseudo-probabilità *p* e *q*, con *p*=a e *q=1-p*:

```
p = (Exp[rf * dt] - d) / (u - d)
0.511915
q = 1 - p
0.488085
```

Il prezzo al tempo *0,* a, e lo strike price, *X,* saranno pari a:

```
s[0] = 50; x = 50;
```

Al fine di ottenere, con *Mathematica*, una rappresentazione a matrice dell'albero dei

prezzi, è necessario procedere nel seguente modo. In primo luogo, occorre definire tutti i prezzi che l'asset può realizzare nel corso dello sviluppo dell'albero. Al massimo il prezzo potrà crescere, al raggiungimento della scadenza dell'opzione, di un ammontare pari a a, dove *per* rappresenta il numero complessivo di stadi dell'albero. Analogamente, il prezzo potrà scendere fino ad un minimo che dipende da  a. La parte "alta" dell'albero sarà costituita dai prezzi che vengono calcolati con un ciclo For:

```
For[i = 1, i <= per, i++,
 s[i] = s[i - 1] * u];
```

La parte "bassa" comprenderà i prezzi calcolati con la stessa procedura ma con indice negativo:

```
For[i = 1, i <= per, i++,
 s[-i] = s[-i + 1] * d];
```

Possiamo ora costruire la matrice che accoglierà l'albero binomiale. Tale matrice ha dimensioni (2 *per+1) x (per+1)*, ma solo una parte verrà effettivamente occupata dai prezzi. Ogni suo elemento è indicato con il simbolo  s[i,j], dove i identifica la riga della matrice (ed il particolare nodo, positivo in caso di crescita del prezzo, negativo viceversa, dell'albero binomiale) e j la colonna (ovvero lo stadio di sviluppo dell'albero). All'inizio, a ciascun elemento viene attribuito un valore nullo (""), per "pulire" la matrice.

```
Table[s[j, k] = "", {j, -per, per}, {k, 0, per}];
```

Con un ciclo di tipo "annidato", si attribuiscono agli specifici elementi della matrice i corrispondenti valori che può assumere il sottostante:

```
For[n = 0, n <= per, n++,
  For[i = 0, i <= n, i++,
    s[2 * i - n, n] = s[2 * i - n]
 ]
]
```

La matrice-albero apparirà quindi in questa forma:

```
AlberoTitolo = MatrixForm[Table[s[-h, k], {h, -per, per}, {k, 0, per}]]
```

$$\begin{pmatrix} & & & & 74.2403 \\ & & & 68.5971 & \\ & & 63.3829 & & 63.3829 \\ & & 58.565 & & 58.565 \\ & 54.1133 & & 54.1133 & & 54.1133 \\ 50 & & 50 & & 50 & \\ & 46.1994 & & 46.1994 & & 46.1994 \\ & & 42.6876 & & 42.6876 & \\ & & 39.4428 & & 39.4428 \\ & & & 36.4447 & \\ & & & & 33.6744 \end{pmatrix}$$

Una volta costruito l'albero dei prezzi, è possibile con una procedura a ritroso calcolare prima il valore dell'opzione a scadenza, quindi per ciascun nodo fino a quello corrente, che indica anche il prezzo stimato dell'opzione.
Costruiamo una matrice che rappresenterà l'albero dei prezzi dell'opzione per ogni nodo, secondo gli stessi schemi esposti in precedenza.

```
Table[eurocall[j, k] = "", {j, -per, per}, {k, 0, per}];
```

A scadenza, il payoff di una call europea *plain-vanilla* è dato da *Max(S-X,0)*. Per ciascun nodo dell'ultimo stadio avremo pertanto:

```
For[i = 0, i <= per, i++,
    eurocall[2 * i - per, per] = Max[s[2 * i - per, per] - x, 0]]
```

Negli stadi antecedenti, fino al primo, il prezzo della call in ogni nodo rifletterà l'aspettativa, attualizzata, dell'opzione basata sui due successivi sviluppi, *up* oppure *down*:

```
For[n = per - 1, n >= 0, n--,
  For[i = n, i >= 0, i--,
    eurocall[2 * i - n, n] =
   Exp[-riskfree * dt] * (p * eurocall[2 * i - n + 1, n + 1] + q * eurocall[2 * i - n - 1, n + 1])
 ]
]
```

E' possibile visualizzare l'albero dell'opzione:

```
AlberoCallEuropea = MatrixForm[Table[eurocall[-h, k], {h, -per, per}, {k, 0, per}]]
```

$$\begin{pmatrix} & & & & 24.2403 \\ & & & 18.8465 & \\ & & 13.8804 & & 13.3829 \\ & 9.74472 & & 8.81435 & \\ 6.57762 & & 5.50721 & & 4.1133 \\ 4.30073 & & 3.32345 & & 2.09516 \\ & 1.95683 & & 1.06719 & 0 \\ & & 0.543587 & 0 & \\ & & 0 & & 0 \\ & & & 0 & \\ & & & & 0 \end{pmatrix}$$

La call europea vale pertanto, in prossimità del nodo [0,0] (cioè correntemente), 4.30073.
Se, ad ogni nodo, anziché attribuire all'opzione direttamente il valore atteso attualizzato dei possibili valori futuri, viene eseguito prima un test sulla possibile convenienza di un esercizio anticipato della medesima, il modello può essere sfruttato per il calcolo di un'opzione americana:

```
Table[amercall[j, k] = "", {j, -per, per}, {k, 0, per}];
For[i = 0, i <= per, i++,
    amercall[2*i - per, per] = Max[s[2*i - per, per] - x, 0]]
For[n = per - 1, n >= 0, n--,
  For[i = n, i >= 0, i--,
    amercall[2*i - n, n] = Max[
    Exp[-riskfree*dt] *
     (p*amercall[2*i - n + 1, n + 1] + q*amercall[2*i - n - 1, n + 1]),
    s[2*i - n, n] - x
    ]
   ]
 ]
```

L'albero risulterà in questo caso dato dai seguenti prezzi:

```
AlberoCallAmericana =
 MatrixForm[Table[amercall[-h, k], {h, -per, per}, {k, 0, per}]]
```

| | | | | |
|---|---|---|---|---|
| | | | | 24.2403 |
| | | | 18.8465 | |
| | | 13.8804 | | 13.3829 |
| | 9.74472 | | 8.81435 | |
| 6.57762 | | 5.50721 | | 4.1133 |
| 4.30073 | 3.32345 | | 2.09516 | |
| | 1.95683 | 1.06719 | | 0 |
| | | 0.543587 | 0 | |
| | | 0 | | 0 |
| | | | 0 | |
| | | | | 0 |

Come mostrato da Hull, nel caso di un'opzione Call non vi è convenienza, qualora il titolo sottostante non paghi alcun dividendo nel corso della vita di essa, all'esercizio anticipato. Le due opzioni hanno dunque lo stesso valore.
Il discorso cambia nel caso di una put. La procedura da seguire per costruire gli alberi è la medesima; è sufficiente modificare soltanto la formula che esprime il payoff a scadenza che, per una put, è pari a *Max(X-S,0)*.

```
Table[europut[j, k] = "", {j, -per, per}, {k, 0, per}];
For[i = 0, i <= per, i++,
    europut[2*i - per, per] = Max[x - s[2*i - per, per], 0]]
For[n = per - 1, n >= 0, n--,
  For[i = n, i >= 0, i--,
    europut[2*i - n, n] =
   Exp[-riskfree*dt] * (p*europut[2*i - n + 1, n + 1] + q*europut[2*i - n - 1, n + 1])
 ]
]
AlberoPutEuropea = MatrixForm[Table[europut[-h, k], {h, -per, per}, {k, 0, per}]]
```

| | | | | |
|---|---|---|---|---|
| | | | | 0 |
| | | | 0 | |
| | | 0 | | 0 |
| | 0.435339 | | 0 | |
| 1.47426 | | 0.896405 | | 0 |
| 3.06622 | 2.57904 | | 1.84578 | |
| | 4.7674 | 4.37032 | | 3.80063 |
| | | 7.11156 | 7.063 | |
| | | | 10.0597 | 10.5572 |
| | | | | 13.306 |
| | | | | 16.3256 |

Nel caso della put americana avremo:

```
Table[amerput[j, k] = "", {j, -per, per}, {k, 0, per}];
For[i = 0, i <= per, i++,
    amerput[2*i - per, per] = Max[x - s[2*i - per, per], 0]]
For[n = per - 1, n >= 0, n--,
  For[i = n, i >= 0, i--,
    amerput[2*i - n, n] = Max[
   Exp[-riskfree*dt] *
     (p*amerput[2*i - n + 1, n + 1] + q*amerput[2*i - n - 1, n + 1]), x - s[2*i - n, n]
    ]
   ]
 ]
AlberoPutAmericana = MatrixForm[Table[amerput[-h, k], {h, -per, per}, {k, 0, per}]]
```

$$\begin{pmatrix}
 & & & & 0 \\
 & & & 0 & \\
 & & 0 & & 0 \\
 & 0.435339 & & 0 & \\
1.50282 & & 0.896405 & & 0 \\
3.16686 & 2.63786 & & 1.84578 & \\
 & 4.94466 & 4.49143 & & 3.80063 \\
 & & 7.41486 & 7.31237 & \\
 & & & 10.5572 & 10.5572 \\
 & & & & 13.5553 \\
 & & & & 16.3256
\end{pmatrix}$$

L'opzione americana vale, in questo caso, più di quella europea (3.16686 contro 3.06622).
Titolo che paga dividendi discreti.
Consideriamo a questo punto l'ipotesi per cui il titolo sottostante stacchi un dividendo di ammontare noto durante la vita dell'opzione. Innanzitutto "puliamo" la memoria dalle variabili precedentemente definite.

```
ClearAll["Global`*"]
```

Reinseriamo le variabili dell'esempio precedente, attribuendo un nuovo nome alla variabile prezzo corrente, per evidenziare come essa ora contenga anche il dividendo che dovrà essere corrisposto successivamente. In questo caso, la volatilità è quella relativa al prezzo dell'asset depurato dei dividendi che dovranno essere pagati.

```
tm = 6/12; per = 5; dt = tm/per; rf = 0.05; vol = 0.25; u = Exp[vol*Sqrt[dt]];
d = 1/u; p = (Exp[rf*dt] - d)/(u - d); q = 1 - p; sd[0] = 50; x = 50;
```

Supponiamo che il dividendo sia pari a 1.80 e che venga staccato fra 3 mesi.

```
div = 1.80; tempodiv = 3 / 12;
```

Sottraendo dal prezzo corrente il valore attuale del dividendo si ottiene:

```
s[0] = sd[0] - div * Exp[-tempodiv * rf]
```
```
48.2224
```

Si può ora procedere alla costruzione dell'albero per i prezzi "depurati" dai dividendi. I nodi si ricongiungono come sempre.

```
For[i = 1, i <= per, i++,
 s[i] = s[i - 1] * u ];
For[i = 1, i <= per, i++,
 s[-i] = s[-i + 1] * d];
Table[s[j, k] = "", {j, -per, per}, {k, 0, per}];
For[n = 0, n <= per, n++,
  For[i = 0, i <= n, i++,
    s[2 * i - n, n] = s[2 * i - n]
 ]
]
AlberoPrezziNetti = MatrixForm[Table[s[-h, k], {h, -per, per}, {k, 0, per}]]
```
```
⎛                                        71.6009⎞
⎜                                66.1583        ⎟
⎜                        61.1294         61.1294⎟
⎜                56.4828         56.4828        ⎟
⎜         52.1894         52.1894         52.1894⎟
⎜48.2224         48.2224         48.2224        ⎟
⎜         44.5568         44.5568         44.5568⎟
⎜                 41.17            41.17        ⎟
⎜                        38.0405         38.0405⎟
⎜                                35.149         ⎟
⎝                                        32.4772⎠
```

Riaggiungendo ai prezzi "netti" dei nodi antecedenti la data di stacco i corrispondenti valori attualizzati di quest'ultimo si ottiene l'albero definitivo che ci consente di calcolare i prezzi dell'opzione.

```
For[n = 0, n <= per, n++,
 For[i = 0, i <= n, i++,
  If[tempodiv > n*dt,
  s[2*i - n, n] =
    s[2*i - n, n] + div* Exp[-(tempodiv - n*dt)*rf], s[2*i - n, n] = s[2*i - n, n]
   ]
 ]
]
AlberoTitolo = MatrixForm[Table[s[-h, k], {h, -per, per}, {k, 0, per}]]
```

| | | | | |
|---|---|---|---|---|
| | | | | 71.6009 |
| | | | 66.1583 | |
| | | 61.1294 | | 61.1294 |
| | 58.2783 | | 56.4828 | |
| 53.976 | | 52.1894 | | 52.1894 |
| 50. | 50.0179 | | 48.2224 | |
| 46.3434 | | 44.5568 | | 44.5568 |
| | 42.9655 | | 41.17 | |
| | | 38.0405 | | 38.0405 |
| | | | 35.149 | |
| | | | | 32.4772 |

L'opzione call europea verrà calcolata secondo la solita procedura a ritroso.

```
Table[eurocall[j, k] = "", {j, -per, per}, {k, 0, per}];
For[i = 0, i <= per, i++,
    eurocall[2*i - per, per] = Max[s[2*i - per, per] - x, 0]]
For[n = per - 1, n >= 0, n--,
  For[i = n, i >= 0, i--,
    eurocall[2*i - n, n] =
   Exp[-rf*dt]*(p*eurocall[2*i - n + 1, n + 1] + q*eurocall[2*i - n - 1, n + 1])
 ]
]
AlberoCallEuropea = MatrixForm[Table[eurocall[-h, k], {h, -per, per}, {k, 0, per}]]
```

| | | | | |
|---|---|---|---|---|
| | | | | 21.6009 |
| | | | 16.4077 | |
| | | 11.627 | | 11.1294 |
| | 7.85071 | | 6.73221 | |
| 5.11508 | | 3.97073 | | 2.18942 |
| 3.24223 | 2.29841 | | 1.11521 | |
| | 1.31124 | 0.568043 | | 0 |
| | | 0.28934 | 0 | |
| | | 0 | | 0 |
| | | | 0 | |
| | | | | 0 |

Nel caso della call americana, ad ogni nodo viene eseguito il test sull'esercizio anticipato.

```
Table[amercall[j, k] = "", {j, -per, per}, {k, 0, per}];
For[i = 0, i <= per, i++,
    amercall[2*i - per, per] = Max[s[2*i - per, per] - x, 0]]
For[n = per - 1, n >= 0, n--,
  For[i = n, i >= 0, i--,
    amercall[2*i - n, n] = Max[
    Exp[-rf*dt] *
     (p*amercall[2*i - n + 1, n + 1] + q*amercall[2*i - n - 1, n + 1]), s[2*i - n, n] - x
    ]
   ]
 ]
AlberoCallAmericana =
 MatrixForm[Table[amercall[-h, k], {h, -per, per}, {k, 0, per}]]
```

| | | | | |
|---|---|---|---|---|
| | | | | 21.6009 |
| | | | 16.4077 | |
| | | 11.627 | | 11.1294 |
| | 8.27834 | | 6.73221 | |
| 5.33289 | | 3.97073 | | 2.18942 |
| 3.35318 | 2.29841 | | 1.11521 | |
| | 1.31124 | 0.568043 | | 0 |
| | | 0.28934 | 0 | |
| | | 0 | | 0 |
| | | | 0 | |
| | | | | 0 |

Si noti come ora, in presenza di dividendi, l'esercizio anticipato dell'opzione può risultare conveniente e le due call non hanno lo stesso valore.
Costruzione di funzioni personalizzate per il modello binomiale.
E' possibile "compattare" l'intera procedura di pricing sopra rappresentata all'interno di una specifica funzione, che permette di calcolare il valore dello strumento derivato sulla base degli input immessi tra parentesi quadre.

Call Europea

```
BECall[tm_, per_, riskfree_, vol_, yield_, stock_, strike_] :=
 Module[{dt, rf, u, d, p, q, s, x, eurocall},
   dt = tm / per; rf = riskfree - yield; u = Exp[vol * Sqrt[dt]]; d = 1 / u;
  p = (Exp[rf * dt] - d) / (u - d); q = 1 - p;
 s[0] = stock;
  x = strike;
 For[i = 1, i <= per, i++,
 s[i] = s[i - 1] * u];
 For[i = 1, i <= per, i++,
 s[-i] = s[-i + 1] * d];
  Table[s[j, k] = "", {j, -per, per}, {k, 0, per}];
  For[n = 0, n <= per, n++,
  For[i = 0, i <= n, i++,
    s[2 * i - n, n] = s[2 * i - n]]];
  Table[eurocall[j, k] = "", {j, -per, per}, {k, 0, per}];
  For[i = 0, i <= per, i++,
    eurocall[2 * i - per, per] = Max[s[2 * i - per, per] - x, 0]];
  For[n = per - 1, n >= 0, n--,
  For[i = n, i >= 0, i--,
    eurocall[2 * i - n, n] =
     Exp[-riskfree * dt] *
      (p * eurocall[2 * i - n + 1, n + 1] + q * eurocall[2 * i - n - 1, n + 1])]];
  eurocall[0, 0]
 ]
```

Call Americana

```
BACall[tm_, per_, riskfree_, vol_, yield_, stock_, strike_] :=
 Module[{dt, rf, u, d, p, q, s, x, amercall},
   dt = tm / per; rf = riskfree - yield; u = Exp[vol * Sqrt[dt]]; d = 1 / u;
  p = (Exp[rf * dt] - d) / (u - d); q = 1 - p;
 s[0] = stock;
  x = strike;
 For[i = 1, i <= per, i++,
 s[i] = s[i - 1] * u];
 For[i = 1, i <= per, i++,
 s[-i] = s[-i + 1] * d];
  Table[s[j, k] = "", {j, -per, per}, {k, 0, per}];
  For[n = 0, n <= per, n++,
  For[i = 0, i <= n, i++,
    s[2 * i - n, n] = s[2 * i - n]]];
 Table[amercall[j, k] = "", {j, -per, per}, {k, 0, per}];
For[i = 0, i <= per, i++,
    amercall[2 * i - per, per] = Max[s[2 * i - per, per] - x, 0]];
For[n = per - 1, n >= 0, n--,
  For[i = n, i >= 0, i--,
    amercall[2 * i - n, n] = Max[
    Exp[-riskfree * dt] *
       (p * amercall[2 * i - n + 1, n + 1] + q * amercall[2 * i - n - 1, n + 1]),
     s[2 * i - n, n] - x
    ]
   ]
 ];
  amercall[0, 0]
 ]
```

Put Europea

```
BEPut[tm_, per_, riskfree_, vol_, yield_, stock_, strike_] :=
 Module[{dt, rf, u, d, p, q, s, x, europut},
   dt = tm / per; rf = riskfree - yield; u = Exp[vol * Sqrt[dt]]; d = 1 / u;
  p = (Exp[rf * dt] - d) / (u - d); q = 1 - p;
 s[0] = stock;
  x = strike;
 For[i = 1, i <= per, i++,
 s[i] = s[i - 1] * u];
 For[i = 1, i <= per, i++,
 s[-i] = s[-i + 1] * d];
  Table[s[j, k] = "", {j, -per, per}, {k, 0, per}];
  For[n = 0, n <= per, n++,
  For[i = 0, i <= n, i++,
    s[2 * i - n, n] = s[2 * i - n]]];
 Table[europut[j, k] = "", {j, -per, per}, {k, 0, per}];
For[i = 0, i <= per, i++,
    europut[2 * i - per, per] = Max[x - s[2 * i - per, per], 0]];
For[n = per - 1, n >= 0, n--,
  For[i = n, i >= 0, i--,
    europut[2 * i - n, n] =
     Exp[-riskfree * dt] *
       (p * europut[2 * i - n + 1, n + 1] + q * europut[2 * i - n - 1, n + 1])]];
  europut[0, 0]
 ]
```

Put Americana

```
BAPut[tm_, per_, riskfree_, vol_, yield_, stock_, strike_] :=
 Module[{dt, rf, u, d, p, q, s, x, amerput},
   dt = tm / per; rf = riskfree - yield; u = Exp[vol * Sqrt[dt]]; d = 1 / u;
  p = (Exp[rf * dt] - d) / (u - d); q = 1 - p;
 s[0] = stock;
  x = strike;
 For[i = 1, i <= per, i++,
 s[i] = s[i - 1] * u];
 For[i = 1, i <= per, i++,
 s[-i] = s[-i + 1] * d];
  Table[s[j, k] = "", {j, -per, per}, {k, 0, per}];
  For[n = 0, n <= per, n++,
  For[i = 0, i <= n, i++,
    s[2 * i - n, n] = s[2 * i - n]]];
Table[amerput[j, k] = "", {j, -per, per}, {k, 0, per}];
For[i = 0, i <= per, i++,
    amerput[2 * i - per, per] = Max[x - s[2 * i - per, per], 0]];
For[n = per - 1, n >= 0, n--,
  For[i = n, i >= 0, i--,
    amerput[2 * i - n, n] = Max[
   Exp[-riskfree * dt] *
       (p * amerput[2 * i - n + 1, n + 1] + q * amerput[2 * i - n - 1, n + 1]), x - s[2 * i - n, n]
    ]
   ]
 ];
  amerput[0, 0]
 ]
```

Si considerino i seguenti esempi di utilizzo delle funzioni introdotte:

```
BECall[0.5, 30, 0.05, 0.56, 0, 50, 50]
8.31855
BACall[0.5, 30, 0.05, 0.56, 0, 50, 50]
8.31855
BEPut[0.5, 30, 0.05, 0.56, 0, 50, 50]
7.08404
BAPut[0.5, 30, 0.05, 0.56, 0, 50, 50]
7.21863
```

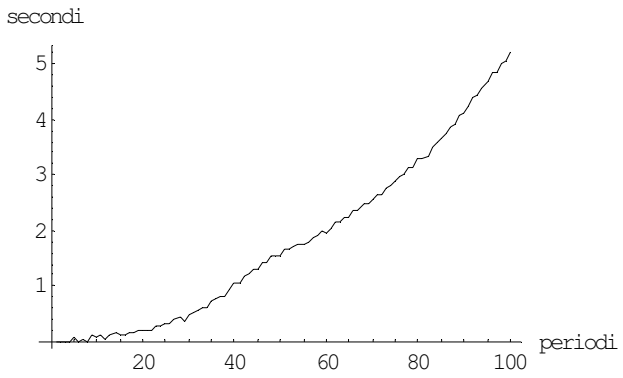
## Velocità e convergenza nel calcolo del modello binomiale

La velocità di calcolo, oltre che essere legata al tipo di processore utilizzato ed alla memoria disponibile, dipende crucialmente dal numero di stadi di sviluppo dell'albero. La funzione interna a *Mathematica* Timing consente di visualizzare i tempi di esecuzione. Consideriamo, ad esempio, la funzione BACall, che calcola il prezzo di una call europea, ipotizzando un numero di stadi che va da 1 a 100. La tabella TempiEPrezzi produce una lista di coppie di valori: il primo elemento rappresenta il tempo necessario per eseguire la formula, ed il secondo rappresenta il valore vero e proprio. Visualizziamo alcuni dei risultati tramite la funzione Short:

```
Short[TempiEPrezzi =
  Table[Timing[BACall[0.5, periodi, 0.05, 0.56, 0, 50, 50]], {periodi, 1, 100}]]
{{0. Second, 10.2688}, <<98>>, {5.22 Second, <<20>>}}
```

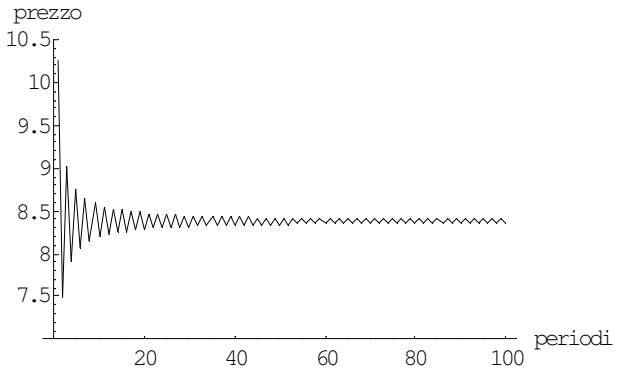
Si noti come i tempi di calcolo richiesti crescano esponenzialmente per una stima più efficiente del prezzo:

```
ListPlot[First[Transpose[TempiEPrezzi]] / Second, PlotJoined -> True,
 AxesLabel -> {"periodi", "secondi"}];
```




La tabella TempiEPrezzi consente anche di studiare il fenomeno della convergenza del modello binomiale verso il giusto prezzo dell'opzione al crescere del numero di stadi dell'albero. La seguente ipotesi riguarda un'opzione at the money.

```
ListPlot[Last[Transpose[TempiEPrezzi]], PlotJoined -> True,
 PlotRange -> {7, 10.5}, AxesLabel -> {"periodi", "prezzo"}];
```




## Calcolo dell'Early Exercise Boundaries delle opzioni americane.

Procediamo ora al calcolo dell'*Early Exercise Boundaries* (EEB) per un'opzione put americana. L'EEB rappresenta il limite di prezzo a partire dal quale diviene conveniente esercitare anticipatamente l'opzione. Nel modello binomiale, ciò significa prendere il prezzo situato in prossimità del nodo dell'albero posto più in alto quando avviene l'esercizio anticipato. Consideriamo una put at the money, con vita residua di sei mesi,una volatilità prevista del 25% ed un tasso d'interesse del 5%. In questo caso i nodi dell'albero vengono posti pari a 150:

```
tm = 6 / 12; per = 150; dt = tm / per; riskfree = 0.05; vol = 0.25; yield = 0;
rf = riskfree - yield; u = Exp[vol * Sqrt[dt]]; d = 1 / u; p = (Exp[rf * dt] - d) / (u - d);
q = 1 - p; s[0] = 50; x = 50;
```

Si procede quindi alla costruzione dell'albero dei prezzi.

```
For[i = 1, i <= per, i++,
 s[i] = s[i - 1] * u];
For[i = 1, i <= per, i++,
 s[-i] = s[-i + 1] * d];
Table[s[j, k] = "", {j, -per, per}, {k, 0, per}];
For[n = 0, n <= per, n++,
  For[i = 0, i <= n, i++,
    s[2 * i - n, n] = s[2 * i - n]
 ]
]
Table[amerput[j, k] = "", {j, -per, per}, {k, 0, per}];
For[i = 0, i <= per, i++,
    amerput[2 * i - per, per] = Max[x - s[2 * i - per, per], 0]]
```

```
For[n = per - 1, n >= 0, n--,
  For[i = n, i >= 0, i--,
    amerput[2 * i - n, n] = Max[
   Exp[-riskfree * dt] *
     (p * amerput[2 * i - n + 1, n + 1] + q * amerput[2 * i - n - 1, n + 1]), x - s[2 * i - n, n]
    ]
   ]
 ]
```

Il prezzo della put americana risulterà pari a:

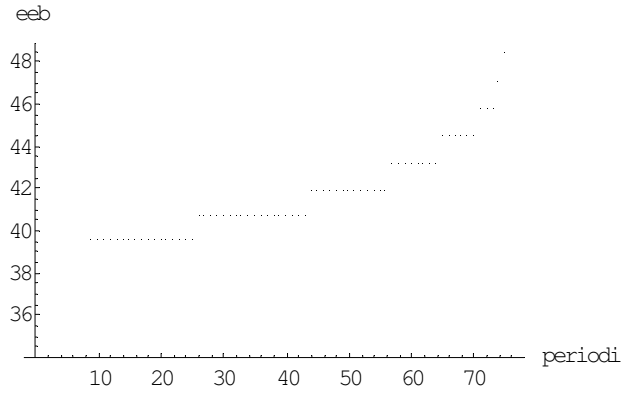
```
amerput[0, 0]
3.00823
```

A questo punto si costruisce una nuova matrice, che verrà destinata ad accogliere, per ciacun nodo dell'albero binomiale, i prezzi in corrispondenza dei quali conviene esercitare anticipatamente l'opzione.

```
Table[exercise[j, k] = 0, {j, -per, per}, {k, 0, per}];
For[k = 0, k <= per, k++,
 For[j = 0, j <= k, j++,
  If[x - s[2 * j - k, k] >
    Exp[-riskfree * dt] * (p * amerput[2 * j - k + 1, k + 1] + q * amerput[2 * j - k - 1, k + 1]),
   exercise[j, k] = s[2 * j - k, k],
   exercise[j, k] = 0]
 ]
]
```

In corrispondenza di ogni passo dell'albero, viene preso il prezzo più alto tra quelli dove si verifica l'esercizio anticipato

```
eeb = Table[Max[Table[exercise[j, k], {j, -per, per}]], {k, 0, per, 2}];
ListPlot[eeb, AxesLabel -> {"periodi", "eeb"}];
```




Distribuzione di probabilità dei rendimenti nell'albero binomiale

Vogliamo verificare adesso come si comporta, al crescere degli stadi dell'albero binomiale, la distribuzione di probabilità dei singoli risultati finali. E' possibile notare che, per un numero di stadi sufficientemente elevato, i prezzi finali si distribuiscano lognormalmente.

In primo luogo, osserviamo come si comportano le probabilità dei singoli nodi.

```
ClearAll["Global`*"]
```

Definiamo le consuete variabili che servono a costruire l'albero:

```
per = 1500; tm = 6 / 12; dt = tm / per; rf = 0.05; vol = 0.25; u = Exp[vol * Sqrt[dt]];
d = 1 / u; p = (Exp[rf * dt] - d) / (u - d); q = 1 - p; stock = 50;
```

Si può scegliere una strada alternativa (oltre che più rapida ) per arrivare ai prezzi finali rispetto a quella finora considerata. In particolare, dopo *i* rialzi e *per-i* ribassi avremo un prezzo finale dato da *S(i)* = a aa.

```
prezzofinale[i_] := stock * u^i * d^(per - i)
```

Per esempio, il prezzo finale corrispondente al nodo centrale dell'albero, con uno stesso numero di rialzi e di ribassi sarà:

```
prezzofinale[750]
50.
```

Corrispondentemente possiamo calcolare il rendimento di periodo:

```
rendfinale[j_] := Log[prezzofinale[j] / stock]
```

Le combinazioni possibili per arrivare ad ogni nodo sono date da $\binom{per}{j}$. Poichè ogni possibile cammino che conduce a quel nodo ha le stesse probabilità di ciascun altro, la probabilità totale di giungere al nodo sarà data da $\binom{\text{per}}{j} p^j q^{\text{per}-j}$.

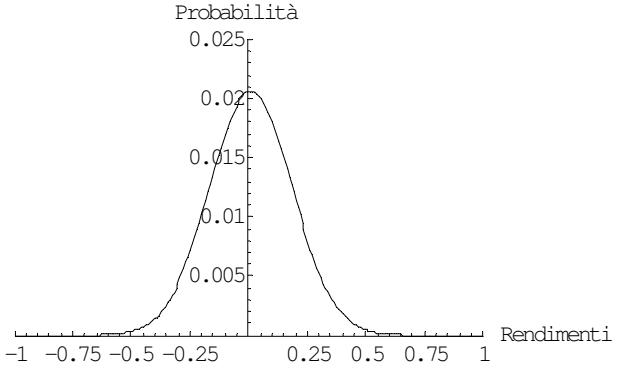
```
prob[j_] := Binomial[per, j] * p^j * q^(per - j)
```

Ciascun rendimento realizzabile viene quindi associato con la sua probabilità.

```
graph = Table[{rendfinale[i], prob[i]}, {i, 0, per}];
```

Graficamente si ottiene:

```
ListPlot[graph, PlotJoined -> True, PlotRange -> {{-1, 1}, {0, 0.025}},
 AxesLabel -> {"Rendimenti", "Probabilità"}];
```




Si noti come le maggiori probabilità si concentrino intorno al valore centrale nullo.
Osserviamo adesso ciò che succede ai prezzi. Quando l'albero si sviluppa per un adeguato numero di livelli, i prezzi tendono a distribuirsi in maniera lognormale ed i risultati tendono a convergere verso quelli della formula di Black & Scholes. Già con 12 stadi si può dimostrare questo:

```
per = 12;
```

Importiamo due *packages* che si riveleranno utili per la rappresentazione finale dei prezzi ottenuti. Questi ultimi, vengono contenuti a tal fine in una lista chiamata prezzi.

```
<< Graphics`Graphics`
<< Statistics`DataManipulation`
prezzi = {};
```

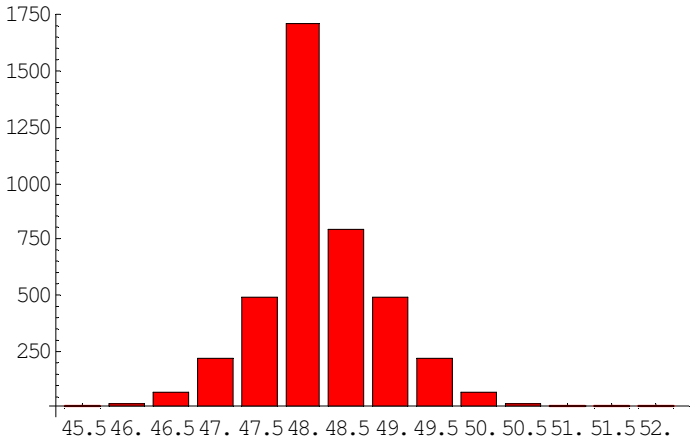
Tale lista contiene tutti i possibili prezzi cui si giunge attraverso ogni cammino che conduce a qualsiasi nodo. Poichè, come precedentemente illustrato, ogni nodo finale *j*

può essere raggiunto a volte, il prezzo corrispondente a quel nodo sarà ripetuto sulla base del coefficiente binomiale.

```
For[j = 1, j <= per, j++,
 For[i = 1, i <= Binomial[per, j], i++,
  AppendTo[prezzi, prezzofinale[j]]
 ]
]
```

Raggruppiamo i prezzi in intervalli di ampiezza 5 e rappresentiamo graficamente la loro distribuzione:

```
prezzofinale[6]
50.
frequenze = BinCounts[prezzi, {47, 54, 0.5}]
{0, 12, 66, 220, 495, 1716, 792, 495, 220, 66, 12, 1, 0, 0}
labels = Table[45 + 0.5 i, {i, 1, 14}]
{45.5, 46., 46.5, 47., 47.5, 48., 48.5, 49., 49.5, 50., 50.5, 51., 51.5, 52.}
BarChart[Transpose[{frequenze, labels}], PlotRange -> All];
```




## Valutazione di opzioni europee con il modello di Black & Scholes

Il modello di Black & Scholes risulta di facile implementazione con *Mathematica*, grazie alla sua forma chiusa. E' necessario, prima di tutto, richiamare il *package* `*ContinousDistribution*`, contenuto nella cartella *Statistics*, di norma già presente al momento dell'installazione tipica di *Mathematica*.

```
ClearAll["Global`*"]
<< Statistics`ContinuousDistributions`
```

E' possibile dunque costruire la funzione ausiliaria *N(*a, che rappresenta la densità

cumulata di una distribuzione normale standardizzata al punto a:

```
enne[d_] := N[CDF[NormalDistribution[0, 1], d]];
```

La definizione delle funzioni per il calcolo di call e put europee è immediata. Il prezzo della call sarà infatti dato da:

```
BSCall[s_, x_, v_, r_, t_] :=
 Module[{d1 = (Log[s / x] + (r + 0.5 v^2) t) / (v Sqrt[t])},
 N[s enne[d1] - x Exp[-r t] enne[d1 - v Sqrt[t]]]
 ];
```

Mentre quello della put sarà pari a:

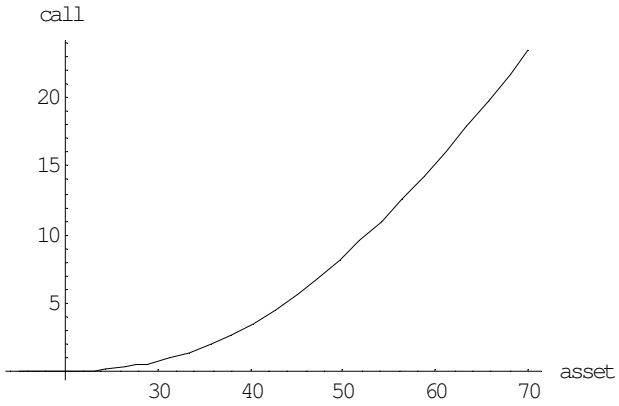
```
BSPut[s_, x_, v_, r_, t_] :=  BSCall[s, x, v, r, t] + x Exp[-r t] - s;
```

Consideriamo alcuni esempi:

```
BSCall[50, 50, 0.56, 0.05, 0.5]
8.38297
BSPut[50, 50, 0.56, 0.05, 0.5]
7.14846
```

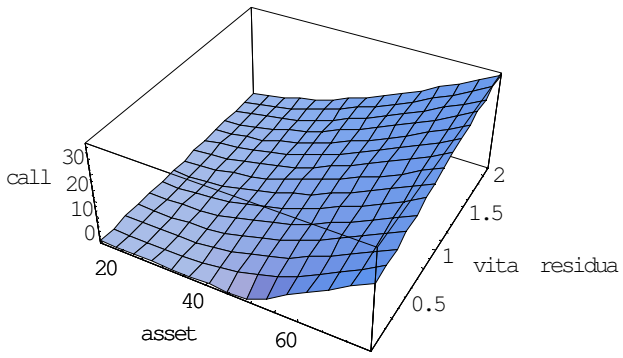
E' possibile visualizzare graficamente le varie relazioni esistenti tra prezzo dell'opzione e input che concorrono alla sua definizione. Ecco mostrato l'andamento del premio dovuto per una call con strike price a 50 per vari livelli di prezzo del sottostante, quando la volatilità è pari al 56%, il tasso d'interesse è del 5% e la vita residua è di 6 mesi:

```
Plot[BSCall[s, 50, 0.56, 0.05, 0.5], {s, 15, 70}, AxesLabel -> {"asset", "call"}];
```




Con il trascorrere del tempo, la curvatura del prezzo si trasforma nella nota *"hockey stick"* che esprime il payoff della call a scadenza:

```
Plot3D[BSCall[s, 50, 0.56, 0.05, t], {s, 15, 75}, {t, 0.001, 2},
 AxesLabel -> {"asset", "vita residua", "call"}];
```




## Costruzione di posizioni complesse.

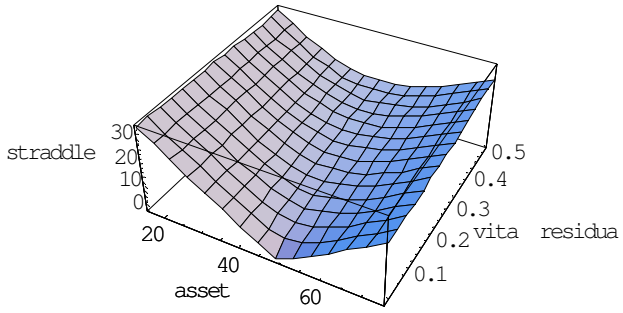
Si possono facilmente costruire strumenti strutturati sfruttando le precedenti funzioni. Si consideri un *long straddle*, ottenuto acquistando una call ed una put at the money, con medesima scadenza.

```
Straddle[s_, x_, v_, r_, t_] := BSCall[s, x, v, r, t] + BSPut[s, x, v, r, t]
Straddle[50, 50, 0.56, 0.05, 0.5]
```

```
15.5314
```

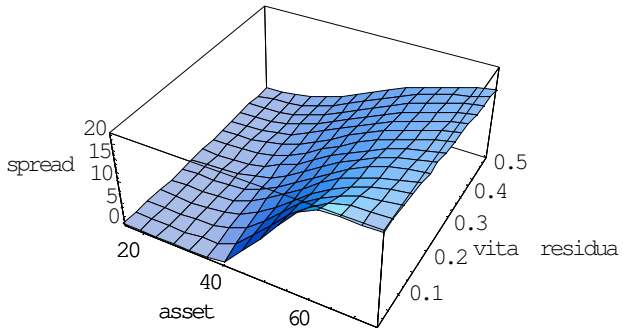
Si può osservare il suo andamento rispetto al prezzo dell'asset ed al tempo.

```
Plot3D[Straddle[s, 50, 0.56, 0.05, t], {s, 15, 75}, {t, 0.001, 0.5},
 AxesLabel -> {"asset", "vita residua", "straddle"}];
```



Consideriamo ora un *call bull spread,* ottenuto combinando l'acquisto di un call in the money con la vendita di un call out of the money. Il suo andamento rispetto al sottostante ed al trascorrere del tempo risulterà in questi termini:

```
Plot3D[BSCall[s, 40, 0.56, 0.05, t] - BSCall[s, 60, 0.56, 0.05, t],
 {s, 15, 75}, {t, 0.00001, 0.5}, AxesLabel -> {"asset", "vita residua", "spread"}];
```




## Analisi della sensitività delle opzioni: i "Greci".

Sfruttando la funzione D[f[x]]che calcola la derivata di una funzione, si possono ottenere con facilità le derivate delle funzioni di call e put rispetto alle variabili che ne influenzano il prezzo; si ottengono i seguenti risultati:

```
RhoCall[s_, x_, v_, r0_, t_] := D[BSCall[s, x, v, r, t], r] /. r -> r0;
RhoPut[s_, x_, v_, r0_, t_] := D[BSPut[s, x, v, r, t], r] /. r -> r0;
VegaCall[s_, x_, v0_, r_, t_] := D[BSCall[s, x, v, r, t], v] /. v -> v0;
VegaPut[s_, x_, v0_, r_, t_] := D[BSPut[s, x, v, r, t], v] /. v -> v0;
DeltaPut[s0_, x_, v_, r_, t_] := D[BSPut[s, x, v, r, t], s] /. s -> s0;
DeltaCall[s0_, x_, v_, r_, t_] := D[BSCall[s, x, v, r, t], s] /. s -> s0;
GammaCall[s0_, x_, v_, r_, t_] := D[DeltaCall[s, x, v, r, t], s] /. s -> s0;
GammaPut[s0_, x_, v_, r_, t_] := D[DeltaPut[s, x, v, r, t], s] /. s -> s0;
ThetaCall[s_, x_, v_, r_, t0_] := -D[BSCall[s, x, v, r, t], t] /. t -> t0;
ThetaPut[s_, x_, v_, r_, t0_] := -D[BSPut[s, x, v, r, t], t] /. t -> t0;
```

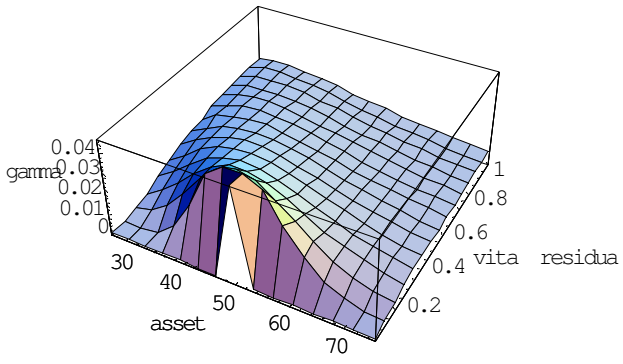
Ecco alcuni esempi numerici:

```
TableForm[Greci = {{delta, DeltaCall[50, 50, 0.56, 0.05, 0.5]}, {gamma,
    GammaCall[50, 50, 0.56, 0.05, 0.5]}, {vega, VegaCall[50, 50, 0.56, 0.05, 0.5]},
   {theta, ThetaCall[50, 50, 0.56, 0.05, 0.5]},
   {rho, RhoCall[50, 50, 0.56, 0.05, 0.5]}},
 TableHeadings -> {None, {"GRECI", "VALORI"}}]
```

| GRECI | VALORI |
|---|---|
| delta | 0.603002 |
| gamma | 0.0194742 |
| vega | 13.632 |
| theta | -8.72226 |
| rho | 10.8836 |

Visualizziamo graficamente il comportamento del gamma (ovvero della sensibilità del delta a mutamenti nel prezzo del sottostante) di una call.

```
Plot3D[GammaCall[q, 50, 0.56, 0.05, z], {q, 25, 75}, {z, 0.0001, 1},
 AxesLabel -> {"asset", "vita residua", "gamma"}];
```




## Calcolo delle volatilità implicite.

La funzione FindRoot può essere efficacemente utilizzata per trovare recursivamente le volatilità implicite delle opzioni.

```
IVCall[s_, x_, r_, t_, c0_] := v /. FindRoot[BSCall[s, x, v, r, t] == c0, {v, 0.01}];
IVPut[s_, x_, r_, t_, p0_] := v /. FindRoot[BSPut[s, x, v, r, t] == p0, {v, 0.01}];
```

Reinserendo i dati degli esempi precedenti, con il prezzo della call che risultava dalla funzione BSCall prima definita, si ottiene esattamente il valore di volatilità prima ipotizzato.

```
IVCall[50, 50, 0.05, 0.5, 8.38297]
```

```
0.56
```

# Valutazione di opzioni esotiche con il metodo Monte-Carlo.

Le opzioni cosiddette "esotiche" si distinguono dalle *plain-vanilla* finora considerate in quanto il loro payoff a scadenza risulta essere più complesso. Quelle che affronteremo in questa sezione sono definite *path-dependent*, poiché il loro valore al momento della scadenza dipende dal cammino percorso dal sottostante nel corso della loro vita. Proprio questa caratteristica rende più problematico il prezzaggio delle medesime, soprattutto se si vuole ricorrere ad una forma chiusa come quella di Black & Scholes, che deve essere modificata proprio per tener conto delle diverse modalità di determinazione del valore a scadenza, rendendo di frequente necessario il ricorso ad approssimazioni. Una soluzione alternativa, che certamente pone problemi in termini di tempi di calcolo superiori e quindi di effettiva applicazione pratica delle medesime, ma che arriva con notevole precisione ai risultati finali, è quella del ricorso a procedure numeriche quali le simulazioni Monte-Carlo e gli alberi binomiali. Il ricorso a questi espedienti diventa addirittura inevitabile quando tali opzioni presentano caratteristiche (come la possibilità di esercizio anticipato) che rendono impossibile l'utilizzo di forme chiuse derivate dalla formula di Black & Scholes.
In questo lavoro verranno analizzate tre diverse tipologie di opzioni esotiche, tutte *path-dependent*:
- opzioni asiatiche;
- opzioni *lookback;*
- opzioni *barrier.*

## Opzioni Asiatiche

Nelle opzioni asiatiche, la definizione del payoff finale passa attraverso la determinazione del prezzo medio registrato nel corso di un certo periodo dal sottostante. Questa media, a seconda dei casi, si sostituisce al prezzo finale dell'asset nell'individuare il valore intrinseco dell'opzione (*Average Rate Asian Option*) oppure determina lo strike price della medesima (*Average Strike Asian Option*).

### Opzioni Asiatiche Europee

#### Average Rate Asian Call Europea

Il payoff a scadenza per questo tipo di opzione sarà dato da $Max\,(Avg(a) - K, 0)$ dove $Avg(a)$ rappresenta la media dei prezzi calcolata per un periodo da aa $T$ e dove $K$ costituisce lo strike price della call. Se consideriamo apari alla data di inizio dell'opzione, la procedura per calcolare il valore di quest'ultima con Monte-Carlo sarà quella che segue. Dopo aver ripulito il sistema dalle variabili precedentemente introdotte ed aver definito la funzione Mean, che permette di calcolare la media:

```
ClearAll["Global`*"]
```

si può procedere alla definizione delle variabili. Il metodo Monte-Carlo prevede la riproduzione del comportamento dell'asset sottostante per un elevato numero di volte. In corrispondenza di ciascuna simulazione viene determinato il valore finale dell'opzione

secondo il payoff definito in precedenza. Il valore corrente dell'opzione sarà pertanto pari alla media dei valori finali ottenuti per ciascuna simulazione, attualizzati ad oggi.
Per riprodurre il cammino del titolo sottostante l'opzione seguiamo la procedura consigliata da Hull: si considera come sempre un albero binomiale con un certo numero di stadi; in corrispondenza di ciascun *step,* il prezzo può intraprendere due direzioni, una al rialzo ed una al ribasso. Data una pseudo-probabilità *p* ottenuta come già analizzato nella prima sezione relativa agli alberi binomiali, si estrae a sorte un numero casuale compreso tra 0 e 1 da una distribuzione uniforme (tramite la funzione Random[ ]): se questo valore è compreso tra 0 e *p*, il prezzo allo stadio successivo crescerà di un fattore *u*; se al contrario è maggiore di *p* (ma minore di 1), il prezzo subirà un ribasso pari a *d*. Questa procedura, che prevede dunque dei salti discreti del prezzo dell'asset, e che per i valori di *u*, *d*, *p* precedentemente definiti corrisponde ad un processo Browniano nel continuo come quello ipotizzato nel mondo di Black & Scholes, viene ripetuta fino al raggiungimento dell'ultimo *step*, e quindi del prezzo del titolo alla scadenza dell'opzione.
A titolo di esempio, si consideri un albero con 30 stadi, e 1000 simulazioni del cammino del prezzo del sottostante. Si ipotizzi inoltre per il titolo una volatilità del 56%, un tasso d'interesse del 5% ed un valore iniziale pari a 50. Le variabili da definire saranno di conseguenza:

```
sim = 1000;
per = 30;
tm = 0.5;
vol = 0.56;
dt = tm / per;
stock = 50;
strike = 50;
u = Exp[vol * Sqrt[dt]];
d = 1 / u;
p = (Exp[rf * dt] - d) / (u - d);
rf = 0.05;
```

Al tempo 0, per ciascuna delle 1000 simulazioni il prezzo dell'asset è uguale a quello iniziale ipotizzato, cioè 50.

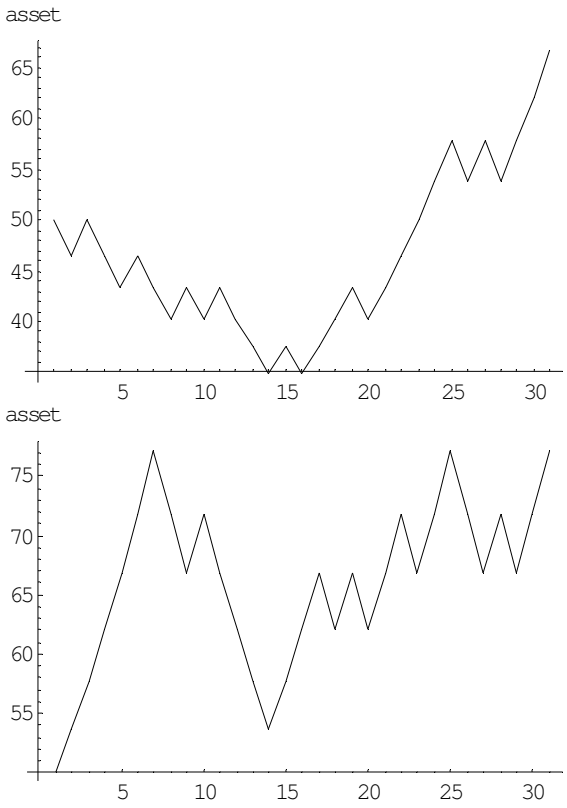
```
Table[s[0, k] = stock, {k, 1, sim}];
```

Si procede dunque con un ciclo For necessario a produrre le simulazioni (in tutto avremo 30 x 1000 possibili prezzi dell'asset).

```
For[j = 1, j <= sim, j++,
 For[i = 1, i <= per, i++,
  Max[0, If[Random[] < p, s[i, j] = s[(i - 1), j] * u, s[i, j] = s[(i - 1), j] * d]]
 ]
]
```

Possiamo visualizzare il *path* percorso dall'asset in due simulazioni, la prima e l'ultima delle 1000 previste:

```
Graphics[ListPlot[Table[s[i, 1], {i, 0, per}], PlotJoined -> True,
  AxesLabel -> {"", "asset"}], ListPlot[Table[s[i, 1000], {i, 0, per}],
  PlotJoined -> True, PlotJoined -> True, AxesLabel -> {"", "asset"}]];
```

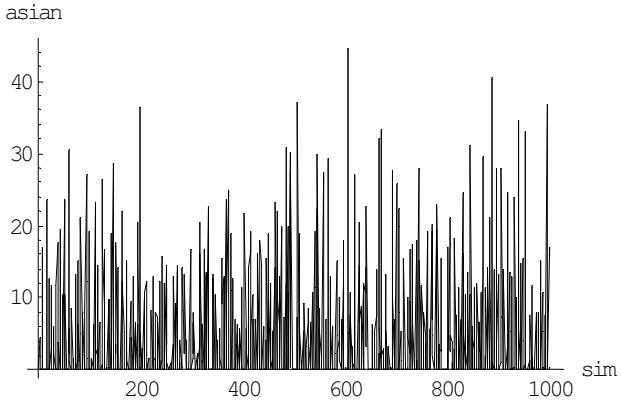



A questo punto si può calcolare il valore finale dell'opzione per ciascuna simulazione. Occorre, a tal fine, calcolare la media dei prezzi registrati in ciascun percorso seguito dal titolo.

```
For[j = 1, j <= sim, j++,
 asian[j] = Exp[-tm * rf] * Max[0, Mean[Table[s[i, j], {i, 0, per}]] - strike]
 ]
```

I prezzi (attualizzati) dell'opzione per le 1000 simulazioni sono visualizzati di seguito:

```
ListPlot[Table[asian[y], {y, 1, sim}], PlotJoined -> True, PlotRange -> All,
 AxesLabel -> {"sim", "asian"}];
```




Il valore finale sarà dato dalla media dei precedenti.

```
AsianAvgRateCall = N[Mean[Table[asian[j], {j, 1, sim}]]]
4.61526
```

Possiamo raggruppare la procedura in un'unica funzione:

```
AsianAverageRateCall[stock_, strike_, tm_, rf_, per_, sim_, vol_] :=
 Module[{dt, d, u, p, price, s, asian},
  dt = tm / per;
  u = Exp[vol * Sqrt[dt]];
  d = 1 / u;
  p = (Exp[rf * dt] - d) / (u - d);
  Table[s[0, k] = stock, {k, 1, sim}];
  For[j = 1, j <= sim, j++,
   For[i = 1, i <= per, i++,
    Max[0, If[Random[ ] < p, s[i, j] = s[(i - 1), j] * u, s[i, j] = s[(i - 1), j] * d]
     ]
    ]
   ];
  For[j = 1, j <= sim, j++,
   asian[j] = Exp[-tm * rf] * Max[0, Mean[Table[s[i, j], {i, 0, per}]] - strike]
   ];
  price = N[Mean[Table[asian[j], {j, 1, sim}]]]
  ]
```

Qualche esempio:

```
AsianAverageRateCall[50, 50, 0.5, 0.05, 30, 10000, 0.56] // Timing
{62.5 Second, 4.80619}
```

Per ottenere dati più attendibili, come si può vedere, è necessario un numero sufficientemente elevato di simulazioni, al prezzo però di un tempo di esecuzione non sempre accettabile.

### *Average Strike Asian Call Europea*

L'altra versione di opzione asiatica prevede che sia lo strike price a dipendere dalla media dei prezzi. Il payoff sarà in questo caso dato da *Max (*a- *Avg(*a*), 0)*, dove a rappresenta il prezzo a scadenza del titolo.
La procedura richiesta per generare i cammini del sottostante è sempre la stessa. Possiamo usare lo stesso insieme di *path* ottenuti in precedenza. Ciò che cambia è naturalmente la formula per il payoff.

```
For[j = 1, j <= sim, j++,
 asianb[j] = Exp[-tm * rf] * Max[0, s[per, j] - Mean[Table[s[i, j], {i, 0, per}]]]
  ]
AsianAvgStrkCall = N[Mean[Table[asianb[j], {j, 1, sim}]]]
4.3507
```

La funzione per calcolare il valore di una *Average Strike Asian Call* sarà data da:

```
AsianAverageStrikeCall[stock_, tm_, rf_, per_, sim_, vol_] :=
 Module[{dt, d, u, price, s, asian},
  dt = tm / per;
  u = Exp[vol * Sqrt[dt]];
  d = 1 / u;
  p = (Exp[rf * dt] - d) / (u - d);
  Table[s[0, k] = stock, {k, 1, sim}];
  For[j = 1, j <= sim, j++,
   For[i = 1, i <= per, i++,
    Max[0, If[Random[] < p, s[i, j] = s[(i - 1), j] * u, s[i, j] = s[(i - 1), j] * d]
     ]
    ]
   ];
  For[j = 1, j <= sim, j++,
   asian[j] = Exp[-tm * rf] * Max[0, s[per, j] - Mean[Table[s[i, j], {i, 0, per}]]]
   ];
  price = N[Mean[Table[asian[j], {j, 1, sim}]]]
  ]
```

Un esempio, ottenuto considerando 10000 simulazioni:

```
AsianAverageStrikeCall[50, 0.5, 0.05, 30, 10000, 0.56] // Timing
{63.82 Second, 4.87159}
```

### *Average Strike Asian Call Americana*

Quando l'opzione asiatica prevede la possibilità di esercizio anticipato, l'uso del metodo Monte-Carlo può non risultare più efficiente, in quanto in ciascuno dei nodi (*per* x *sim* nell'esempio) andrebbe testata la convenienza a tale esercizio. Si può ricorrere all'albero binomiale, che peraltro non mostra un sensibile miglioramento da un punto di vista dei tempi di esecuzione. Sfruttando le funzioni recursive di *Mathematica*, si possono costruire "al volo" delle liste di prezzi che rappresentano ciascuno dei percorsi che il prezzo del titolo può seguire e, arrivando a scadenza, calcolare il payoff finale dell'opzione, che viene poi riportato indietro fino al momento della valutazione (sempre tenendo conto in ciascun nodo del possibile esercizio anticipato). Il tutto può essere rappresentato in maniera molto sintetica tramite la seguente funzione:

```
AsianAverageStrikeCallAmerican[stock_, tm_, rf_, per_, vol_] :=
 Module[{dt = tm / per, d, u, p, q},
  u = Exp[vol Sqrt[dt]];
  d = 1 / u;
  p = (Exp[rf dt] - d) / (u - d);
  q = 1 - p;
  asian[price_List, j_] :=
   If[j == per, Max[0, Last[price] - Mean[price]],
    Exp[-rf dt] Max[
      Max[0, Last[price] - Mean[price]],
      p asian[Append[price, Last[price] u], j + 1] +
       q asian[Append[price, Last[price] d], j + 1]
       ]
    ];
    asian[{stock}, 0]
   ]
```

La scelta di un elevato numero di stadi dell'albero, come già accennato, porta a dei problemi di attesa, dato che i possibili cammini del prezzo dell'asset crescono esponenzialmente all'aumentare degli *steps*.

```
AsianAverageStrikeCallAmerican[50, 0.5, 0.05, 15, 0.5] // Timing
{22.3 Second, 5.39144}
```

Si noti come, per semplicità, sono state trattate soltanto opzioni per le quali il periodo di monitoraggio del prezzo (necessario per il calcolo della media) coincideva completamente con la vita dell'opzione. Si è in particolare ipotizzato nell'ambito delle simulazioni che, in prossimità di ciascun nodo, venissero osservati i prezzi. Inoltre, le seguenti procedure sono valide solo allorché applicate ad un'opzione che parte al momento della valutazione. In caso contrario, occorrerebbe tener conto dei prezzi finora registrati dall'asset, ai fini della giusta determinazione della media a scadenza. La considerazione di quest'ultimo fattore, peraltro, non mostra particolari problemi, dato che la media finora registrata dal prezzo può costituire un input aggiuntivo facilmente introducibile nella funzione.

## Opzioni Lookback

Le opzioni *lookback* permettono di sfruttare al pieno le evoluzioni (se favorevoli) del sottostante, attribuendo al *buyer* la possibilità:
- di acquistare (vendere in caso di put) il titolo al minor (maggior) prezzo registrato nel corso di un periodo prefissato - *Floating Strike Lookback Call (Put);*
- di incassare, quando positiva, la differenza massima tra il prezzo registrato dal titolo ed uno strike predefinito per una call (ovvero la differenza positiva tra lo strike ed il minimo prezzo prodotto dal titolo nel caso di una put) - *Fixed Strike Lookback Call (Put).*

La procedura è la medesima utilizzata per le opzioni asiatiche: è sufficiente semplicemente modificare i payoff.

### Floating Strike Lookback Call Europea

```
LookbackFloatingStrikeCall[stock_, tm_, rf_, per_, sim_, vol_] :=
 Module[{dt, d, u, price, s, lkbk},
  dt = tm / per;
  u = Exp[vol * Sqrt[dt]];
  d = 1 / u;
  p = (Exp[rf * dt] - d) / (u - d);
  Table[s[0, k] = stock, {k, 1, sim}];
  For[j = 1, j <= sim, j++,
   For[i = 1, i <= per, i++,
    Max[0, If[Random[ ] < p, s[i, j] = s[(i - 1), j] * u, s[i, j] = s[(i - 1), j] * d]
     ]
    ]
   ];
  For[j = 1, j <= sim, j++,
   lkbk[j] = Exp[-tm * rf] * Max[0, s[per, j] - Min[Table[s[i, j], {i, 0, per}]]]
   ];
  price = N[Mean[Table[lkbk[j], {j, 1, sim}]]]
  ]
```

Ecco un esempio

```
LookbackFloatingStrikeCall[50, 0.5, 0.05, 30, 10000, 0.5] // Timing
{66.96 Second, 12.216}
```

## Fixed Strike Lookback Call Europea

```
LookbackFixedStrikeCall[stock_, strike_, tm_, rf_, per_, sim_, vol_] :=
 Module[{dt, d, u, price, s, lkbk},
  dt = tm / per;
  u = Exp[vol * Sqrt[dt]];
  d = 1 / u;
  p = (Exp[rf * dt] - d) / (u - d);
  Table[s[0, k] = stock, {k, 1, sim}];
  For[j = 1, j <= sim, j++,
   For[i = 1, i <= per, i++,
    Max[0, If[Random[ ] < p, s[i, j] = s[(i - 1), j] * u, s[i, j] = s[(i - 1), j] * d]
     ]
    ]
   ];
  For[j = 1, j <= sim, j++,
   lkbk[j] = Exp[-tm * rf] * Max[0, Max[Table[s[i, j], {i, 0, per}]] - strike]
   ];
  price = N[Mean[Table[lkbk[j], {j, 1, sim}]]]
  ]
LookbackFixedStrikeCall[50, 50, 0.5, 0.05, 30, 10000, 0.5] // Timing
{64.98 Second, 14.464}
```

### Floating Strike Lookback Call Americana

Nel caso di opzioni americane, si può ricorrere sempre al binomiale:

```
LookbackFloatingStrikeCallAmerican[stock_, tm_, rf_, per_, vol_] :=
 Module[{dt = tm / per, d, u, p, q},
  u = Exp[vol Sqrt[dt]];
  d = 1 / u;
  p = (Exp[rf dt] - d) / (u - d);
  q = 1 - p;
  lkbk[price_List, j_] :=
   If[j == per, Max[0, Last[price] - Min[price]],
    Exp[-rf dt] Max[
      Max[0, Last[price] - Min[price]],
      p lkbk[Append[price, Last[price] u], j + 1] +
       q lkbk[Append[price, Last[price] d], j + 1]
       ]
    ];
    lkbk[{stock}, 0]
   ]
LookbackFloatingStrikeCallAmerican[50, 0.5, 0.05, 16, 0.5] // Timing
{47.34 Second, 11.5936}
```

### Opzioni Barrier

Le opzioni *barrier* possono essere semplici opzioni alle quali viene posto un vincolo aggiuntivo, sulla base del quale esse entrano effettivamente in vita, o cessano di esistere, a seconda che il prezzo del sottostante tocchi determinati *boundaries.*
Si distingue pertanto tra opzioni *"knock-in"* e opzioni *"knock-out"*. A seconda poi che il limite venga intaccato dall'alto oppure dal basso, si parla di opzioni *"down and in"* o *"down and out"* e di *"up and in"* e *"up and out"*.

### *Barrier Call Europea "knock-in"*

Ripuliamo la memoria dalle variabili esistenti e ridefiniamo la funzione per il calcolo della media.

```
ClearAll["Global`*"]
```

Consideriamo una call europea at the money che preveda, oltre alle consuete variabili, anche un limite superiore, upbound, pari a 65, che definisce il limite il quale, una volta superato, fa entrare in vita l'opzione. Poiché il prezzo iniziale dell'asset è 50, l'opzione è di tipo *up and in*.

```
sim = 100;
stock = 50;
strike = 50;
rf = 0.05;
tm = 0.5;
vol = 0.5;
per = 30;
dt = tm / per;
u = Exp[vol * Sqrt[dt]];
d = 1 / u;
p = (Exp[rf * dt] - d) / (u - d);
upbound = 65;
```

Si procede alla consueta generazione di cammini casuali dell'asset.

```
Table[s[0, k] = stock, {k, 1, sim}];
For[j = 1, j <= sim, j++,
  For[i = 1, i <= per, i++,
   Max[0, If[Random[] < p, s[i, j] = s[(i - 1), j] * u, s[i, j] = s[(i - 1), j] * d]
   ]
  ]
 ];
```

Il payoff della call dipende dalla circostanza per cui il limite sia stato raggiunto. Se il prezzo massimo raggiunto all'interno di ciascuna simulazione raggiunge upbound, l'opzione entra in vita, e varrà il proprio valore intrinseco, quando positivo.

```
For[j = 1, j <= sim, j++,
  If[Max[Table[s[i, j], {i, 0, per}]] > upbound,
  barr[j] = Exp[-tm * rf] * Max[0, -strike + s[per, j]], barr[j] = 0
  ]
 ];
```

Il prezzo dell'opzione è dato, come di consueto, dalla media dei valori di ciascuna simulazione:

```
price = N[Mean[Table[barr[j], {j, 1, sim}]]]
10.7398
```

Riscriviamo il tutto come funzione autonoma:

```
BarrierCallKI[stock_, strike_, upbound_, tm_, rf_, per_, sim_, vol_] :=
 Module[{dt, d, p, u, price, s, barrier},
 dt = tm / per;
 u = Exp[vol * Sqrt[dt]];
 d = 1 / u;
 p = (Exp[rf * dt] - d) / (u - d);
 Table[s[0, k] = stock, {k, 1, sim}];
 For[j = 1, j <= sim, j++,
  For[i = 1, i <= per, i++,
   Max[0, If[Random[ ] < p, s[i, j] = s[(i - 1), j] * u, s[i, j] = s[(i - 1), j] * d]
   ]
  ]
 ];
 For[j = 1, j <= sim, j++,
  If[Max[Table[s[i, j], {i, 0, per}]] > upbound,
  barrier[j] = Exp[-tm * rf] * Max[0, -strike + s[per, j]], barrier[j] = 0
  ]
 ];
 price = N[Mean[Table[barrier[j], {j, 1, sim}]]]
 ]
```

*Barrier Call Europea "knock-out"*

La stessa procedura può essere seguita per opzioni call *knock-out*. Si noti come, in questo caso, l'opzione cessa di esistere se il limite viene raggiunto. La funzione potrà essere scritta come:

```
BarrierCallKO[stock_, strike_, upbound_, tm_, rf_, per_, sim_, vol_] :=
 Module[{dt, d, p, u, price, s, barrier},
 dt = tm / per;
 u = Exp[vol * Sqrt[dt]];
 d = 1 / u;
 p = (Exp[rf * dt] - d) / (u - d);
 Table[s[0, k] = stock, {k, 1, sim}];
 For[j = 1, j <= sim, j++,
  For[i = 1, i <= per, i++,
   Max[0, If[Random[ ] < p, s[i, j] = s[(i - 1), j] * u, s[i, j] = s[(i - 1), j] * d]
   ]
  ]
 ];
 For[j = 1, j <= sim, j++,
  If[Max[Table[s[i, j], {i, 0, per}]] > upbound, barrier[j] = 0,
  barrier[j] = Exp[-tm * rf] * Max[0, -strike + s[per, j]]
  ]
 ];
 price = N[Mean[Table[barrier[j], {j, 1, sim}]]]
 ]
```

*Barrier Put Europea "knock-in" e "knock-out"*

Per le opzioni put il discorso viene ribaltato: considerando un limite inferiore, lobound, otteniamo le formule per opzioni knock-in e knock-out:

```
BarrierPutKI[stock_, strike_, lobound_, tm_, rf_, per_, sim_, vol_] :=
 Module[{dt, d, p, u, price, s, barrier},
 dt = tm / per;
 u = Exp[vol * Sqrt[dt]];
 d = 1 / u;
 p = (Exp[rf * dt] - d) / (u - d);
 Table[s[0, k] = stock, {k, 1, sim}];
 For[j = 1, j <= sim, j++,
  For[i = 1, i <= per, i++,
   Max[0, If[Random[ ] < p, s[i, j] = s[(i - 1), j] * u, s[i, j] = s[(i - 1), j] * d]
   ]
  ]
 ];
 For[j = 1, j <= sim, j++,
  If[Min[Table[s[i, j], {i, 0, per}]] < lobound,
  barrier[j] = Exp[-tm * rf] * Max[0, strike - s[per, j]], barrier[j] = 0
  ]
 ];
 price = N[Mean[Table[barrier[j], {j, 1, sim}]]]
 ]
```

```
BarrierPutKO[stock_, strike_, lobound_, tm_, rf_, per_, sim_, vol_] :=
 Module[{dt, d, p, u, price, s, barrier},
 dt = tm / per;
 u = Exp[vol * Sqrt[dt]];
 d = 1 / u;
 p = (Exp[rf * dt] - d) / (u - d);
 Table[s[0, k] = stock, {k, 1, sim}];
 For[j = 1, j <= sim, j++,
  For[i = 1, i <= per, i++,
   Max[0, If[Random[ ] < p, s[i, j] = s[(i - 1), j] * u, s[i, j] = s[(i - 1), j] * d]
   ]
  ]
 ];
 For[j = 1, j <= sim, j++,
  If[Min[Table[s[i, j], {i, 0, per}]] < lobound, barrier[j] = 0,
  barrier[j] = Exp[-tm * rf] * Max[0, strike - s[per, j]]
  ]
 ];
 price = N[Mean[Table[barrier[j], {j, 1, sim}]]]
 ]
```

Concludiamo con alcuni esempi di utilizzo delle formule:

```
BarrierCallKI[50, 50, 75, 0.5, 0.05, 30, 1000, 0.5]
5.33174
BarrierCallKO[50, 50, 75, 0.5, 0.05, 30, 1000, 0.5]
2.52237
BarrierPutKI[50, 50, 35, 0.5, 0.05, 30, 1000, 0.5]
4.2984
BarrierPutKO[50, 50, 35, 0.5, 0.05, 30, 1000, 0.5]
1.32355
```

## Bibliografia


Benninga S., Steinmetz R., Stroughair J., *Implementing Numerical Option Pricing Models*, The Mathematica Journal, 3-4 1993.

Black F., Scholes M., *The Pricing Of Options And Corporate Liabilities*, Journal Of Political Economy, 81 1973.

Clewlow L., Strickland C. (ed.), *Exotic Options, The State Of The Art*, International Thomson Business Press, 1997.

Cox J.C., Ross S.A., Rubinstein M., *Option Pricing: A Simplified Approach*, Journal Of Financial Economics, 7 1979.
Hull J., *Options, Futures And Other Derivative Securities*, Prentice Hall, 1993.
Miller R.M., *Option Valuation*, in Varian H., *Economic And Financial Modeling With Mathematica*, Telos, 1992.

# Appendice

## Costruzione di un albero binario (con nodi che non si ricongiungono)

In questa appendice viene mostrato come è possibile giungere alla determinazione di un albero dei prezzi nel quale, in prossimità di ogni nodo, i due sentieri successivi che l'asset può intraprendere non sono comuni a quelli di ciascun altro nodo dello stesso livello, ovvero alla realizzazione di una struttura ad albero binaria, senza ricongiungimento dei nodi. Per prima cosa, definiamo le consuete variabili, ipotizzando solo 4 stadi di sviluppo dell'albero.

```
ClearAll["Global`*"]
per = 4; rf = 0.05; tm = 6/12; dt = tm/per; vol = 0.56; u = Exp[vol*Sqrt[dt]];
d = 1/u; s[1, 0] = 50; pos[1, 0] = 0;
```

Definiamo la variabile p, designata a contenere i prezzi nelle varie fasi di sviluppo dell'albero:

```
Table[p[i, k] = "", {i, -2^per - 1, 2^per - 1}, {k, 0, per}];
```

In ogni stadio k  dell'albero, ciascun nodo viene numerato da 1 fino all'ultimo, per un numero complessivo pari a a di nodi. Da ogni nodo, al livello successivo il prezzo può crescere (e raggiungere un nodo pari) oppure scendere, verso un nodo dispari. Per ogni valore che può assumere il nodo viene pertanto svolto un test, tramite la funzione IntegerQ, al fine di verificare se esso è pari, ed atto quindi ad accogliere un prezzo in crescita, oppure dispari, per ottenere un prezzo in diminuzione. Tali possibili prezzi vengono inseriti nella variabile temporanea s:

```
For[k = 1, k <= per, k++,
 For[i = 1, i <= 2^k, i++,
  If[IntegerQ[i/2] == True,
   s[i, k] = s[i/2, k - 1] u,
   s[i, k] = s[(i + 1)/2, k - 1] d
   ]
 ]
]
```

Dopo il primo stadio il prezzo può scendere da 50 a 50 *d* (nodo dispari, 1)

```
s[1, 1]
41.0189
```

Oppure crescere a 50 *u* (nodo pari, 2)

```
s[2, 1]
60.9475
```

Occorre adesso preparare la matrice destinata ad accogliere i prezzi nella rappresentazione dell'albero. A tal scopo, ciascuna posizione rilevante nella matrice viene individuata tramite la funzione pos.

```
For[k = 1, k <= per, k++,
 For[i = 1, i <= 2^k, i++,
  If[IntegerQ[i / 2] == True,
   pos[i, k] = pos[i / 2, k - 1] + 2^(per - k),
   pos[i, k] = pos[(i + 1) / 2, k - 1] - 2^(per - k)
     ]
 ]
]
```

Il punto di partenza sarà individuato dagli indici 1 e 0, e si troverà presso la riga centrale della matrice:

```
pos[1, 0]
0
```

Da quel punto, considerando solo 4 stadi, le posizioni *up* e *down* sono date rispettivamente da:

```
pos[1, 1]
-8
```

e da:

```
pos[2, 1]
8
```

Inserendo nella matrice il prezzo iniziale ed attribuendo una posizione ai prezzi successivi che possono essere realizzati nel modello, otteniamo:

```
p[1, 0] = s[1, 0]
50
For[k = 1, k <= per, k++,
 For[i = 1, i <= 2^k, i++,
  p[pos[i, k], k] = s[i, k]
  ]
]
```

Il prezzo *up* dello stadio successivo a quello iniziale avrà la posizione:

```
p[8, 1]
60.9475
```

Se indichiamo una posizione estranea agli sviluppi dell'albero non otteniamo nessun valore:

```
p[5, 1]
```

Possiamo quindi visualizzare l'albero binario così ottenuto:

```
Table[p[-i, k], {i, -(2^per - 1), 2^per - 1}, {k, 0, per}] // MatrixForm
```

```
⎛                          110.386⎞
⎜                  90.5582        ⎟
⎜                           74.292⎟
⎜           74.292                ⎟
⎜                           74.292⎟
⎜                  60.9475        ⎟
⎜                            50.  ⎟
⎜   60.9475                       ⎟
⎜                           74.292⎟
⎜                  60.9475        ⎟
⎜                            50.  ⎟
⎜             50.                 ⎟
⎜                            50.  ⎟
⎜                  41.0189        ⎟
⎜50                         33.651⎟
⎜                                 ⎟
⎜                           74.292⎟
⎜                  60.9475        ⎟
⎜                            50.  ⎟
⎜             50.                 ⎟
⎜                            50.  ⎟
⎜                  41.0189        ⎟
⎜                           33.651⎟
⎜   41.0189                       ⎟
⎜                            50.  ⎟
⎜                  41.0189        ⎟
⎜                           33.651⎟
⎜           33.651                ⎟
⎜                           33.651⎟
⎜                  27.6066        ⎟
⎝                          22.6478⎠
```

Si può facilmente osservare che, per le assunzioni fatte in merito all'andamento dell'azione (che varia sulla base di un fattore moltiplicativo), i nodi in realtà si ricongiungono, nonostante questa forma di rappresentazione, perchè ad esempio s[2,2]=s[1,0] d u che è esattamente la stessa cosa di s[3,2]=s[1,0] u d. Tuttavia, la ramificazione completa dell'albero torna utile in caso di *path dependance*, come ad esempio nelle opzioni asiatiche, dove all'albero del prezzo si può affiancare quello della media osservata, nel quale i nodi non si ricongiungono. Seguendo la stessa logica sopra rpesentata otteniamo:

```
Table[mp[i, k] = "", {i, -2^per - 1, 2^per - 1}, {k, 0, per}];
```

Al primo stadio, il prezzo medio coincide con quello iniziale.

```
mean[1, 0] = s[1, 0];
```

Nei livelli successivi, la media viene aggiornata con il nuovo prezzo registrato.

```
For[k = 1, k <= per, k++,
 For[i = 1, i <= 2^k, i++,
  If[IntegerQ[i / 2] == True,
   mean[i, k] = (mean[i / 2, k - 1] * (k) + s[i / 2, k - 1] u) / (k + 1),
   mean[i, k] = (mean[(i + 1) / 2, k - 1] * (k) + s[(i + 1) / 2, k - 1] d) / (k + 1)
   ]
 ]
]
```

Il nuovo albero che contiene i prezzi medi accoglierà la variabile mp.

```
mp[1, 0] = mean[1, 0]
50
For[k = 1, k <= per, k++,
 For[i = 1, i <= 2^k, i++,
  mp[pos[i, k], k] = mean[i, k]
  ]
]
```

Rappresentando graficamente la struttura si ottiene:

```
Table[mp[-i, k], {i, -(2^per - 1), 2^per - 1}, {k, 0, per}] // MatrixForm
```

```
⎛                          77.2367⎞
⎜                  68.9494        ⎟
⎜                          70.0179⎟
⎜          61.7465                ⎟
⎜                          64.0958⎟
⎜                  61.5467        ⎟
⎜                          59.2374⎟
⎜  55.4738                        ⎟
⎜                          59.2374⎟
⎜                  55.4738        ⎟
⎜                           54.379⎟
⎜          53.6492                ⎟
⎜                          50.3933⎟
⎜                  50.4916        ⎟
⎜50                        47.1235⎟
⎜                                 ⎟
⎜                          55.2517⎟
⎜                  50.4916        ⎟
⎜                          50.3933⎟
⎜          47.0063                ⎟
⎜                          46.4076⎟
⎜                  45.5095        ⎟
⎜                          43.1378⎟
⎜  45.5095                        ⎟
⎜                          43.1378⎟
⎜                  41.4222        ⎟
⎜                           39.868⎟
⎜          41.5566                ⎟
⎜                          37.1855⎟
⎜                  38.0691        ⎟
⎝                          34.9849⎠
```

Si noti come in questo caso non necessariamente i nodi vengano a ricongiungersi, in quanto la media dipende crucialmente dal percorso finora seguito dal prezzo per arrivare ad un dato punto.

Questo approccio si rivela tuttavia dispendioso da un punto di vista dei tempi di calcolo, dato che il numero di nodi cresce in maniera esponenziale all'aumentare dei livelli (al livello $k$ si avranno anodi totali). Un albero con soli 30 stadi presenterà ben 1073741824 nodi!

*Maurizio Spadaccino*