Commento al programma per il calcolo del prezzo di un'opzione e della relativa copertura

Riccardo Rossi

15/06/2005

1 Teoria

Nel programma a cui fa riferimento questo testo, simuleremo la traiettoria del prezzo di un'opzione (in particolare, di una Call) tramite il modello binomiale Cox, Ross e Rubinstein.

Sia S_n il valore del sottostante al tempo n, allora:

$$S_{n+1} = \begin{cases} S_n(1+a) & \text{con probabilità } p \\ \\ S_n(1+b) & \text{con probabilità } (1-p) \end{cases} \tag{1}$$

con $a, b \in \mathbb{R}$.

Affinché il modello sia privo di arbitraggio, occorre p tale che soddisfi:

$$p(1+a) + (1-p)(1+b) = 1 + r \tag{2}$$

dove r è l'interesse per l'investimento non rischioso.

L'unica soluzione dell'equazione (2) è

$$p = \frac{b-r}{b-a}$$

la quale determina l'unica misura di martingala equivalente. Questo ci garantisce la completezza del mercato e quindi l'esistenza di una strategia replicante per ogni opzione.

Ora, dai parametri dell'opzione dobbiamo determinarne il prezzo tramite il metodo deterministico e indicare una sua strategia replicante, ovvero esplicitare una strategia ϕ tale che il valore del portafoglio alla scadenza sia pari alla funzione Payoff dell'opzione. In formule:

$$V_N(\phi) = \phi_N^0 S_N^0 + \phi_N S_N = (S_N - K)_+;$$

e, essendo $c(n, S_n)$ la funzione di prezzo dell'opzione al tempo n, deve valere per ogni n:

$$\phi_n^0 S_n^0 + \phi_n S_n = c(n, S_n) \tag{3}$$

Operativamente, simuleremo una traiettoria del valore del sottostante tramite il modello CRR e, per ogni generico istante $n-1$, determineremo le quantità ϕ_n, ϕ_n^0 rispettivamente del titolo sottostante e del titolo non rischioso da acquistare al tempo n per garantire la copertura dal rischio.

Osserveremo inoltre che, pur facendo evolvere il valore di S_n con una probabilità $q \in (0,1)$ qualsiasi, da sostituire alla p nella (1), il programma determina comunque una strategia replicante valida, per la quale, alla scadenza, avremo che il valore del portafoglio uguaglierà il payoff dell'opzione.

Per sottolineare questo fatto, prenderemo a caso $q \in (0,1)$ ad ogni lancio

del programma e verificheremo che, effettivamente, la differenza tra valore del portafoglio e il payoff dell'opzione sarà nulla.

Dalla teoria, sappiamo che valgono le seguenti formule. Sia $F(x)$ la funzione di Payoff, allora:

$$c(n, S_n) = (1+r)^{-(N-n)} \left[\sum_{j=0}^{N-n} \binom{N-n}{j} p^j (1-p)^{N-n-j} F(S_n(1+a)^j (1+b)^{N-n-j}) \right] \tag{4}$$

$$\phi_n = \frac{c(n, S_{n-1}(1+b)) - c(n, S_{n-1}(1+a))}{S_{n-1}(b-a)} \tag{5}$$

$$\phi_n^0 = (1+r)^{-n} \Big(c(n, S_{n-1}(1+a)) - \frac{1+a}{b-a} \big(c(n, S_{n-1}(1+b)) - c(n, S_{n-1}(1+a)) \big) \Big) \tag{6}$$

2 Spiegazione del codice

Nel codice sorgente del programma, definiamo per primi i parametri del problema quali ad esempio volatilità e maturità, e in seguito alcune variabili di lavoro.

Abbiamo definito dei vettori di numeri reali di lunghezza $N+1$ in modo da tenere in memoria le quantità possedute e i valori sia del sottostante che del titolo non rischioso. Nel codice, abbiamo chiamato $S_n^0, \phi_n^0, S_n, \phi_n$ rispettivamente So[n], Fo[n], S[n], F[n].

Scegliamo prima la misura di probabilità neutrale al rischio, assegnando $p = \frac{b-r}{b-a}$, poi determiniamo $q \in (0,1)$, probabilità di evoluzione del valore del sottostante, numero casuale ottenuto usando la funzione rand() di C.

A questo punto entriamo nel cuore del programma.

All'interno di un ciclo for, calcoliamo al generico istante i, facendo riferimento alle (5) e (6), le quantità ϕ_{i+1} e ϕ^0_{i+1} che consentono la copertura dal rischio. Solo in un secondo momento facciamo evolvere il prezzo del sottostante a S_{n+1}.

Se i calcoli fatti sono corretti, deve valere l'uguaglianza (3); quindi per verifica, ad ogni iterazione, sommiamo il modulo della differenza tra valore del portafoglio al tempo n e rispettivo prezzo dell'opzione.

A ciclo finito, stampiamo sia la somma degli eventuali errori tra valore del portafoglio e prezzo dell'opzione accumulati durante le iterazioni, sia il modulo della differenza tra valore del portafoglio alla scadenza e il payoff dell'opzione.

Se il programma viene eseguito senza errori e rispettando le previsioni, viene stampato un messaggio di conferma.

3 Codice Sorgente in C

```
/*
Questo programma consente di calcolare il prezzo e la copertura per un'opzione
Call di tipo Europeo, la cui evoluzione viene simulata con il modello di Cox,
Ross e Rubinstein.
Se si volesse considerare una Put invece che una Call basterebbe cambiare la
funzione di Payoff.
```

```
Autore: Riccardo Rossi
*/

#include <stdlib.h>
#include <math.h>
#include <time.h>

double payoff(double, double);
double binomiale(int, int);
double costo(int, double, double, int, double, double, double, double);


int main() {

/* Definiamo i dati del problema caratterizzanti l'opzione */
const double sigma = 0.2;
const double R = 0.05;
const double T = 1.;
const int N = 360;

/* Definiamo le variabili del modello di CCR */
double r = R / N;
double a = (1 + r) * exp(-sigma * sqrt(T/N)) - 1; // per definizioe
```

```
double b = (1 + r) * exp(sigma * sqrt(T/N)) - 1; // per definizione
double So[N+1];
double Fo[N+1];
double F[N+1];
double S[N+1];
S[0] = 100.; // Prezzo iniziale del sottostante



/* Definiamo alcune variabili di lavoro */
srand( (unsigned)time( NULL ));
int i, j;
for (i = 0; i <= N; i++) So[i] = pow(1+r, i); // titolo non rischioso
double errore = 0; // variabile che sommera' gli eventuali errori parziali


double pstar = (b - r) / (b - a);
double p; // Generica


/* Prezziamo l'opzione al tempo 0 */
printf("\nIl prezzo della CALL al tempo 0 \ncon maturita' T = %f anno/i,\n"
    "prezzo iniziale del sottostante So = %f Euro \n"
    "tasso istantaneo R = %f (per cento) annuo,\nvolatilita' %f,
    \nmonitoraggio N = %d \ne' pari a %f Euro.\n\n", T, S[0], R,
    sigma, N, costo(0, S[0], r, N, pstar, a, b, S[0]));
```

```
/* Scegliamo una probabilita' casuale di discesa del prezzo */
p = (double)rand()/RAND_MAX;

for (i = 1; i <= N; i++) {

    /* Strategia replicante */
    F[i] = (costo(i, S[i-1]*(1+b), r, N, pstar, a, b, S[0]) -
    costo(i, S[i-1]*(1+a), r, N, pstar, a, b, S[0])) / (S[i-1]*(b-a));
    Fo[i] = pow(1+r, -i)*(costo(i, S[i-1]*(1+a),r,N,pstar, a, b, S[0]) -
    F[i]*S[i-1]*(1+a));

    /* Facciamo evolvere il prezzo del sottostante */
    if ( (double)rand() / RAND_MAX < p) S[i] = S[i-1]*(1 + a);
    else S[i] = S[i-1]*(1 + b);

    /* Sommiamo gli errori ad ogni iterazione */
    errore += fabs(S[i]*F[i] + So[i]*Fo[i] -
    costo(i, S[i], r, N, pstar, a, b, S[0]));
}

/* Verifichiamo che l'errore sia "nullo", ovvero che abbiamo
calcolato una valida strategia replicante */
printf("L'errore tra valore del portafoglio e il prezzo dell'opzione "
    "sommato ad ogni iterazione \ne' pari a %f. \n", fabs(errore) );
```

```
printf("Inoltre l'errore tra valore del portafoglio alla maturita' e il "
    "payoff dell'opzione \ne' pari a %f. \n", fabs(Fo[N]*So[N] +
    F[N]*S[N] - payoff(S[N], S[0])) );

if ( fabs(Fo[N]*So[N] + F[N]*S[N] - payoff(S[N], S[0])) < 1e-6 )
    printf("Prova Riuscita \n");
else printf("Prova Fallita \n");

return 0;
}


/* Funzione di Payoff per una CALL */
double payoff(double x, double k) {
if (x-k > 0) return x-k;
else return 0;
}

/* Calcolo del coefficiente binomiale */
double binomiale(int n, int k) {
if (k == 0) return 1;
else return (double) ((n-k+1) * binomiale(n, k-1)) / k ;
}
```

```
/* Calcolo della funzione di costo dell'opzione */
double costo(int n, double x, double r, int N, double p, double a,
    double b, double datozero) {
    int j; double ris = 0;
for (j = 0; j <= N-n; j++)
{
    ris += (double)binomiale(N-n, j) * pow(p,j) * pow(1-p,N-n-j) *
        payoff(x * pow(1+a,j) * pow(1+b,N-n-j), datozero);
}
    return ris * pow(1+r,-N+n);
}
```

4 Bibliografia

L. Caramellino, *Introduzione alla finanza.* Dispense del corso di PR5, (2005).