Pricer d’options

Programmation orientée objet

ENSAE PARISTECH

13 février 2015
Créé par : Brian Whannou

Pricer d'options

Programmation orientée objet

I. Présentation générale des méthodes de pricing utilisées

1. Approche intuitive

Une option est un instrument financier qui donne le droit et non l'obligation de recevoir un flux futur. Le montant de ce flux futur dépend d'un sous-jacent (actions, obligations, commodities, aléas climatiques) dont l'évolution exacte n'est pas connu d'avance. La dépendance entre le montant reçu et le sous-jacent est le payoff de l'option. L'exercice de ce droit ne peut pas se faire au-delà d'une certaine date appelée maturité de l'option. Par ailleurs une option que l'on peut exercer avant la maturité (respectivement uniquement à la maturité) est une option américaine (respectivement européenne).

Penchons-nous sur une option européenne. Puisque l'évolution du sous-jacent est aléatoire ou du moins inconnue, l'acheteur d'une option ne connait pas le montant de son payoff à la maturité donc il ne peut se reposer que sur son espérance. Ainsi il prévoit obtenir, à la maturité, l'espérance du payoff. Par conséquent la valeur du droit que confère l'option est la valeur actualisée de l'espérance du payoff. Ecris sous forme mathématique :

$$P = DF(T, R) * E(f(S_T))$$

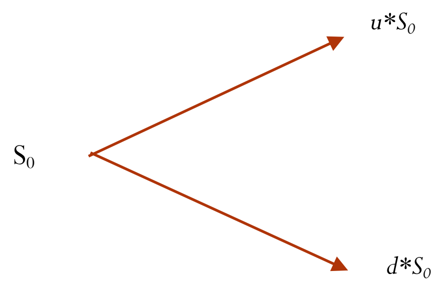
Où DF est le facteur d'actualisation qui dépend de la maturité T et du taux d'intèrêt R, f est le payoff et S_T est la valeur du sous-jacent à la date T. Le principal problème dans cette expression, donc dans l'évaluation du prix, est le calcul de l'espérance car on a l'expression explicite du payoff et du facteur d'actualisation.

2. Calcul de l'espérance

Le calcul de l'espérance revient à faire des hypothèses sur l'évolution du sous-jacent.

a. Le modèle Binomial

Sous ce modèle on doit discrétiser le temps, le temps est une succession de période et à une période donnée le sous-jacent ne peut prendre que deux valeurs.



Le sous-jacent S est multiplié par un facteur u (« up factor » dans le programme) dans le cas favorable et par un facteur d(« down » dans le programme) dans le cas défavorable.

La théorie montre que l'espérance est alors :

$$E\big(f(S_1)\big) = \frac{u-(1+R)}{u-d} * f(d*S_0) + \frac{1+R-d}{u-d} * f(u*S_0)$$

b. Le Modèle de Black-Scholes.

Dans ce modèle, l'évolution de la valeur du sous-jacent est modélisée par un mouvement brownien.

$$S_T = S_0 * \exp(\Big(R - \frac{1}{2}\sigma^2\Big)T + \sigma\sqrt{T}N(0,1))$$

Le problème au niveau de l'implémentation est la simulation de la loi normale.

II. Analyse des différentes classes du programme.

En découpant notre première expression, on a construit différentes classes qui nous permettent de structurer le programme.

1. Facteur d'actualisation.

NOM : DiscountFactor
Méthodes • DiscountFactor(double,double) • ~DisscountFactor() ; • GetMaturity,SetMaturity • GetRate,SetRate • double DF()const=0 ; • DiscountFactor* Clone()const=0 ;
Attributs (protected) • doubleMaturity ; • double Rate ;

Nom : ActuarialDF
Méthodes • ActuarialDF(double,double) ; • ActuarialDF(constActuarialDF&) ; • ~Actuarial() • GetMaturity(),SetMaturity (double) • GetRate(),SetRate(double) • Double DF()const ; • ActuarialDF* Clone()const ;

Nom : ContinuousDF
Méthodes • ContinuousDF(double,double) ; • ContinuousDF(constContinuousDF&) ; • ~Continuous() • GetMaturity(),SetMaturity (double) • GetRate(),SetRate(double) • Double DF()const ; • ContinuousDF* Clone()const ;

ActuarialDF et ContinuousDF sont des classes filles de DiscountFactor .

La classe ActuarialDF actualise avec le facteur $\frac{1}{(1+R)^T}$ alors que ContinuousDF actualise avec le facteur e^{-R*T}

2. L'évolution du sous-jacent

La classe PriceModel est une classe abstraite. De plus pour un besoin de clarté, on a construit deux autres classes abstraites qui sont filles de PriceModel :DiscreteTime et ContinuousTime. Chacune d'elle engendre une classe . Binomial pour la première et BlackS pour la deuxième.

Nom : BlackS
• BlackS(double V, doubleFirstSpot,DiscountFactor&) • Virtual ~BlackS() ; • GetVolatility(),SetVolatility(double) ; • GetFirstSpot(),SetFirstSpot(double) ; • BlackSoperator= (constBlackS&) ; • GetStdE(),SetStdE(double) ; • GetOneStockValue(UniGen&) ; • GetPayOffValue(Option&,UniGen&) ; • PriceMersenne(Option&,int,intseed) ; • PriceMersenne(Option&,double,intseed) ;
Attributs • DoubleVolatility ; • DoubleFirstSpot ; • DiscountFactor* MyDFPtr ; • DoubleStdE ;

Nom : Binomial
Méthodes : • Binomial(doubleup,doubledown ,doublefirstSpot,constDiscountFactor&) ; • Les getters et les setters ; • RiskNeutralProb() ; • Std ::queue<double>Graph()const ; • Std ::queue<double>OptionGraph(constOption&)const ; • Virtual doubleprice(constOption&)
Attributs : • Double down ; • Double up ; • DoublefirstSpot ; • DiscountFactor*MyDFPtr ;

3. Générateur aléatoire

En ce qui concerne les générateurs aléatoires, on a voulu avoir une meilleur fonction que celle fournie par le logiciel à savoir Rand(), donc nous avons pris un programme déjà élaboré sur internet qui génère des variables aléatoires uniforme selon la méthode de mersenne (randomc.h, mersenne.cpp) . Ensuite on a appliqué l'algorithme de Box-Müller pour la transformation en loi normale.

4. La classe PayOff et la classe Option

Les deux classes sont liées l'une étant l'attribut de l'autre. On a séparé cest deux objets pour faciliter l'ajout d'autre type de PayOff.

5. L'interface

Enfin la classe interface gère la liaison humain-machine, s'assurant que l'utilisateur entre les données espérées.

III. Conclusion

En conclusion, ce projet de programmation nous a permis d'aborder des champs aussi divers et variés que la simulation aléatoire, l'utilisation des pointeurs, la gestion de la mémoire avec l'utilisation de la fonction **new** et l'utilisation des classes abstraites comme moyen de généralisation. Malgré la flexibilité de notre programme, on aurait aimé implémenter aussi un pricer pour les options asiatiques et bermudiennes. Par ailleurs une interface graphique avec l'utilisation de la bibliothèque Qt a été un plus envisageable.