Modèles stochastiques en finance MAP 552
École Polytechnique
Département de Mathématiques Appliquées

TP 1.1
Le modèle de Cox-Ross-Rubinstein

Ce TP est associé au Chapitre 2 du polycopié de cours. Lisez-le afin de procéder à l'implémentation demandée.

Une partie des points sera accordée à la présence de commentaires et à la bonne vectorisation du code sur au moins une dimension.

1. Considérons le modèle binomial à $n \geq 1$ périodes avec les coefficients d'évolution :

$$u_n := e^{\mu h_n + \sigma\sqrt{h_n}},$$
$$d_n := e^{\mu h_n - \sigma\sqrt{h_n}},$$

où $\mu > 0, \sigma > 0$ et $h_n := \frac{T}{n}$ pour tout $n \geq 1$ avec $T > 0$ la maturité. On note S_j^n le vecteur de $\mathbb{R}^{j+1}$ des prix possibles à la date $1 \leq j \leq n$ défini par : $S_j^n(i) := S_0 u^{j-i} d^i$ pour $0 \leq i \leq j$ et $S_0 > 0$.

   (a) Écrire une fonction `Sn(T, n, μ, σ, j)` qui renvoie le vecteur S_j^n.

   (b) Considérons une option européenne de type *call* de maturité $T > 0$ et de prix d'exercice $K > 0$. Écrire une fonction `Payoff(T, n, μ, σ, K)` qui renvoie le vecteur des profils de gain à la maturité.

   (c) Soit $r \geq 0$ le taux d'intérêt sans risque (et constant). Écrire une fonction `Calln(T, n, r, μ, σ, K)` qui renvoie le prix de l'option européenne de type *call* ((option d'achat)) à la date 0.

   (d) À chaque date $t_j^n := jh_n$, on note Δ_j^n la stratégie de couverture. Écrire une fonction `Deltan(T, n, r, μ, σ, K, j)` qui renvoie le vecteur Δ_j^n.

   (e) Étudiez la dépendance du prix d'exercice K sur les résultats des fonctions `Calln` et `Deltan`, et commentez. Vous pouvez, par exemple, tester $K \in \{80, 81, \ldots, 120\}$ sur un arbre à $n = 50$ périodes et avec les paramètres

$$\sigma = 30\,\%, \quad r = 5\,\%, \quad \mu = 10\,\%, \quad S_0 = 100, \quad T = 2. \tag{1}$$

2. Nous comparons à la formule de Black Scholes (limite quand le nombre de périodes du modèle binomial tend vers l'infini).

   (a) Écrire une fonction `Call(T, r, σ, K)` qui renvoie le prix à la date 0 d'une option européenne de type *call* avec la formule de Black-Scholes. Pour la fonction de répartition de la loi normale en Python, on pourra utiliser la fonction `scipy.stats.norm.cdf` de la bibliothèque `scipy`.

   (b) Définir la fonction :

`error(T, n, r, μ, σ, K) := Calln(T, n, r, μ, σ, K)/Call(T, r, σ, K) - 1`,

puis tracer le graphique de cette erreur relative en fonction du nombre de périodes n et commentez. Vous pouvez utiliser les paramètres de (1) avec $K = 105$.