A rendre pour le jeudi 28 janvier en TD de processus en finance, de manière individuelle

Exercice : Calcule des prix d'options à l'aide d'un arbre binomial

Consignes

Sont à rendre: un petit rapport contenant les quelques démonstrations mathématiques demandées, les résultats numériques et les codes sources COMMENTÉS. Une archive des sources est à envoyer par mail à `temam@math.jussieu.fr` avec comme sujet `TP1 C++`.

Partie A: Classes des payoff

On introduit une classe (dans un fichier `payoff.h`) décrivant les différents payoff auxquels nous aurons à faire, dont le prototype est
(pour récupérer ce prototype, vous devez copier le fichier `/share/thot/users/enseignement/etemam/TP1/payoff.h`)

```
class PayOff
{
    double Strike; //Prix d'exercice
    double Expiry; // Maturité
    int TheOptionType; // Call=1, Put=2
    int IsAmerican; //Option Américaine=1 Européenne=0
  public:
    PayOff(double Strike, double Expiry, int TheOptionType, int IsAmerican);
    double Evaluate(double Spot) const; //Evaluation du payoff au point Spot
    ~PayOff();
};
```

1. Ecrire le constructeur de cette classe ainsi que la fonction `PayOff::Evaluate`.
2. Vérifier à l'aide d'un programme rapide que vos fonctions sont opérationnelles.
3. On veut remplacer l'entier `TheOptionType` par une liste de nom. Réécrire la classe en utilisant le type personnalisé

   ```
   enum OptionType {call, put};
   ```

4. Etendre la classe aux options digitales (call et put).
5. Modifier la classe en rajoutant des fonctions permettant d'accéder au caractère américain de l'option et à la maturité.
6. **Surcharge d'opérateur:** remplacer la fonction `PayOff::Evaluate` par une surcharge de l'opérateur `()` permettant d'utiliser la classe payoff avec les lignes suivantes:

   ```
   PayOff ThePayOff(OneStrike, PayOff::putdigital);
   double thisPayOff=ThePayOff(OneSpot);
   cout << "Le payoff du putdigital vaut "<< thisPayOff << endl;
   ```

   ou

   ```
   OptionType PutDigital=putdigital;
   PayOff ThePayOff(OneStrike, PutDigital);
   double thisPayOff=ThePayOff(OneSpot);
   cout << "Le payoff du putdigital vaut "<< thisPayOff << endl;
   ```

7. On désire pricer des options avec plusieurs prix d'exercices (call/put spread, collar....). Cela nécessite de modifier la classe en:

   ```
   class PayOff
   {
       double *Strike; //Prix d'exercice
       int NbStrike;
       ....
   };
   ```

Faire les modifications nécessaires sur la classe (attention aux constructeur/destructeur).

8. Faut-il effectuer des modifications à la classe pour que les lignes suivantes soient correctes?

```
double Mat=1.0; double K[2]={100.,110.};
PayOff TheFirstPayOff(Mat,callspread,false,2,K);
PayOff TheSecondPayOff(TheFirstPayOff);
OptionType PutSpread=putspread;
TheSecondPayOff.SetOptionType(PutSpread);
double thisPayOff= TheSecondPayOff(100.);
```

ou

```
double Mat=1.0; double K[2]={100.,110.};
PayOff TheFirstPayOff(Mat,PayOff::callspread,false,2,K);
PayOff TheSecondPayOff(TheFirstPayOff);
TheSecondPayOff.SetOptionType(PayOff::putspread);
double thisPayOff= TheSecondPayOff(100.);
```

Partie B: Les arbres

Les méthodes de pricing appelées "arbre" repose sur le modèle discret de Cox-Ross-Rubinstein. Si on suppose que l'actif sous-jacent du marché satisfait l'équation différentielle stochastique de Black et Scholes:

$$dS_t = S_t(rdt + \sigma dW_t),$$

la valeur d'une option européenne de payoff $f(S_T)$ est alors

$$e^{-rT}\mathbb{E}[f(S_T)].$$

Lorsque que l'on price avec un arbre on divise l'intervalle de temps $[0,T]$ en intervalles de taille constance $\frac{1}{N}$,

$$[0,T] = \left[t_0 = 0, t_1 = \frac{T}{N}\right] \bigcup \ldots \bigcup \left[t_k = \frac{kT}{N}, t_{k+1} = \frac{(k+1)T}{N}\right] \bigcup \left[t_{N-1} = \frac{(N-1)T}{N}, t_N = T\right]$$

A chaque pas de temps, le mouvement brownien évolue suivant l'incrément $W_{t_{k+1}} - W_{t_k}$, que l'on doit approcher. Or cette variable suit une loi normale $\mathcal{N}\left(0, \frac{T}{N}\right)$. L'idée est de remplacer le mouvement brownien par une marche aléatoire, plus simplement, chaque incrément sera approché par une variable de Bernouilli à valeurs dans $\{-1,1\}$ équiprobable. Ainsi, si X_k représente ces variables W_{t_k} est approché par:

$$Y_k = \sqrt{\frac{T}{N}} \sum_{j=1}^{k} X_j.$$

L'algorithme se décompose en deux phases:

- Une phase dite "construction de l'arbre" où l'on part de la valeur initiale de l'actif S_0 et on génère l'arbre à l'aide de la transition
$$x \to x^{\pm} \quad \text{où } x^{\pm} = x\exp\left((r - \sigma^2/2)\frac{T}{N} \pm \sigma\sqrt{\frac{T}{N}}\right).$$
- Une phase de "descente" où l'on calcule le prix à l'aide de la relation trouvée à la question II.2.

1. Vérifier que Y_k et W_{t_k} ont les mêmes deux premiers moments et montrer que quand $N \to +\infty$ Y_N converge vers W_T en loi.
2. En utilisant le caractère martingale de la valeur de l'option, établir une récurrence descendante permettant de calculer le prix approché de l'option.
3. Montrer que pour une option américaine de payoff f, le prix Π^f aux dates t_k satisfait:
$$\begin{aligned} \Pi^f_{t_N}(S_{t_N}) &= f(S_{t_N}) \\ \Pi^f_{t_k}(S_{t_k}) &= \max\left(f(S_{t_k}), \frac{1}{2}\left(\Pi^f_{t_{k+1}}(S^+_{t_k}) + \Pi^f_{t_{k+1}}(S^-_{t_k})\right)\right) \end{aligned}$$

   On s'intéresse maintenant à l'implémentation en C++ de ces méthodes. Le prototype choisit est le suivant: (`/share/thot/users/enseignement/etemam/TP1/tree.h`)

```
class BinomialTree
{
        double R;
        double Vol;
        double Spot;
        int Steps;
        double CurrentTime;
        bool TreeBuilt;
        std::vector<std::vector<double> >  CurrentSpot; //Utilise la STL
        std::vector<std::vector<double> >  CurrentOptionValue; //Utilise la STL
public:
        BinomialTree(
                double R_i=0.1, double Vol_i=0.2, double Spot_i=100.,
                int Steps_i=10, double CurrentTime_i=0.,   bool TreeBuilt_i=false
                );
        void BuiltTree(double T);
        double GetPrice(const PayOff& ThePayOff);
};
```

4. Ecrire le constructeur de cette classe, en utilisant la fonction `resize` de la STL.
5. Ecrire les fonctions `BuiltTree` et `GetPrice` (Attention à la prise en charge des options américaines).
6. Programmer une (et une seule) fonction donnant le prix exact du call, du put et de l'option digitale sous un modèle de Black-Scholes exact.
   Utilisez la fonction `/share/thot/users/enseignement/etemam/TP1/normale.cpp` pour la fonction de répartition de la loi normale.
7. Testez l'algorithme avec divers options et paramètres. A quelle vitesse converge t-il en fonction du nombre de pas (`Steps`) pour les options européennes à payoff continu (on indiquera la vitesse numérique et comment celle-ci aura été calculée)? Européenne à payoff discontinu? Américaine?
8. Sans écrire le code indiquez comment vous utiliseriez cette algorithme pour obtenir la couverture.