DAUPHINE
UNIVERSITÉ PARIS

Master Domaine MIDO - 2ème année

Mention Mathématiques de la Modélisation et de la Décision Spécialité Ingénierie Statistique et Financière, parcours en apprentissage

TP de C++ pour la Finance

2009-2010

Maude Manouvrier

Je tiens à remercier Julien Saunier qui m'a permis de reprendre ses sujets de TP dont certains exercices de ce document en sont largement inspirés, Julien s'étant lui-même largement inspiré des documents de Bernt Arne Ødegaard et de Wikipedia. Certaines parties de ce document sont également inspirées de l'ouvrage "Exercices en langage C++" de Claude Delannoy, Eyrolles, 2004, du cours de Marc Csernel, que j'avais suivi lorsque j'étais étudiante (en 1995), et des documents que m'ont confiés Béatrice Berard et Frédéric Darguesse. Merci à eux !

1 TP1 : Manipulation des types de base et création de fonctions

Pour ce TP, deux bibliothèques C++ sont nécessaires : `cstdlib` et `cmath`.

`cstdlib` (voir documentation à l'adresse `http ://www.cplusplus.com/reference/clibrary/cstdlib/`) qui contient les fonctions suivantes :

```
void srand ( unsigned int seed ) ; // initialisation de la graine du générateur
int rand ( void ) ; // retourne un nombre aléatoire entre 0 et RAND_MAX
```

`cmath` (voir documentation à l'adresse `http ://www.cplusplus.com/reference/clibrary/cmath/`) qui contient les fonctions suivantes :

```
double exp ( double x ) ;
double log ( double x ) ;
double pow ( double x, double y ) ;
double sqrt ( double x ) ;
```

Exercice 1 : **Générer des nombre aléatoires**

1. Ecrire la fonction permettant de renvoyer un nombre aléatoire compris entre 0 et 1.
2. Quel est le problème des générateurs aléatoires, et notamment de celui proposé dans la cstdlib ?
3. Créer une fonction *double* $n(double)$ calculant la loi de distribution normale (ou gaussienne) centrée réduite $n(x) = \frac{1}{\sqrt{2\Pi}} e^{-\frac{x^2}{2}}$.
4. Ecrire la fonction permettant de renvoyer un nombre normal aléatoire -tiré selon une distribution normale uniforme- par la méthode de la forme polaire de Box-Muller (plus rapide et robuste que la forme littérale de la transformation).
   La description algorithmique est la suivante :
   On tire deux nombres au hasard x_0 et x_1 entre 0 et 1.
   On calcule v_0 et v_1 tels que $v_i = 2x_i - 1$.
   Soit $s = v_0^2 + v_1^2$;
   Si s est supérieur à 1, alors on recommence au début.
   Sinon, on récupère $v_0\sqrt{\frac{-2ln(s)}{s}}$.[1]

Exercice 2 : **Black Scholes**

La formule de Black-Scholes permet de calculer la valeur théorique d'une option à partir des cinq données suivantes : S_T la valeur actuelle de l'action sous-jacente à la date T ; $T - t$ le temps qui reste à l'option avant son échéance (exprimé en années), t étant la date courante ; K le prix d'exercice fixé par l'option ; r le taux d'intérêt sans risque ; σ la volatilité du prix de l'action ;
Le prix théorique d'une option d'achat (*call*), qui donne le droit mais pas l'obligation d'acheter l'actif S à la valeur K à la date T, est caractérisé par son "pay off" : $(\mathcal{S}_T - K)^+ = \max(S_T - K; 0)$
Soit la formule de Black-Scholes : $C(S, K, r, T, \sigma) = S\mathcal{N}(d_1) - Ke^{-r(T-t)}\mathcal{N}(d_2)$
De même,le prix théorique d'une option de vente ("put") $(K - \mathcal{S}_T)^+ = \max(K - S_T; 0)$ est donné par :
$P(S, K, r, T, \sigma) = -S\mathcal{N}(-d_1) + Ke^{-r(T-t)}\mathcal{N}(-d_2)$
avec $\mathcal{N}$ la fonction de répartition de la loi normale centrée réduite $\mathcal{N}(0, 1)$ et
$d_1 = \frac{1}{\sigma\sqrt{(T-t)}} \left[\ln\left(\frac{S}{K}\right) + \left(r + \frac{1}{2}\sigma^2\right)(T - t)\right]$
$d_2 = d_1 - \sigma\sqrt{(T-t)}$

En prenant la fonction `N` définie de la manière suivante (en utilisant l'approximation de Abromowitz and Stegunir - cf. `http ://www.math.sfu.ca/~cbm/aands/page32.htm` algo 26.2.17) :

$$N(x) = 1 - n(x)(b_1 t + b_2 t^2 + b_3 t^3 + b_4 t^4 + b_5 t^5) + \epsilon$$

$$t = \frac{1}{1 + 0.2316419x}$$

Soit en C++ :

```
// Approximation de la fonction de distribution cumulative normale (par des séries de Taylor)
// Fonction reprise du document de Bernt Arne Ødegaard
double N(double z)
{
  if (z > 6.0)
    return 1.0;  // éviter les valeurs illicites
  if (z < -6.0)
    return 0.0;
  double b1 = 0.31938153;
  double b2 = -0.356563782;
  double b3 = 1.781477937;
```

[1] Il est à noter que $v_1\sqrt{\frac{-2ln(s)}{s}}$ est aussi un nombre valable.

```
  double b4 = -1.821255978;
  double b5 = 1.330274429;
  double p = 0.2316419;
  double c2 = 0.3989423;
  double a=fabs(z);
  double t = 1.0/(1.0+a*p);
  double b = c2*exp((-z)*(z/2.0));
  double n = ((((b5*t+b4)*t+b3)*t+b2)*t+b1)*t;
  n = 1.0-b*n;
  if ( z < 0.0 )
    n = 1.0 - n;
  return n;
};
```

1. Ecrire la fonction calculant le prix d'un call d'une option ; ainsi que celle calculant le prix d'un put.
2. Ecrire la fonction d'appel de ces deux fonctions pour un action valant actuellement 50, un temps à échéance de 6 mois, une volatilité de l'action estimée à 30%, un taux d'intérêt sans risque de 10%.

   Il est parfois intéressant de calculer les dérivées partielles du prix d'option, le delta et le gamma étant liés aux prix, theta lié au temps, vega à la volatilité et rho au taux d'intérêt. Leurs formules sont les suivantes :

   Delta (Δ)

   $$\Delta = \frac{\partial c}{\partial S} = N(d_1)$$

   Gamma (Γ)

   $$\frac{\partial^2 c}{\partial S^2} = \frac{n(d_1)}{S\sigma\sqrt{T-t}}$$

   Theta (Θ)

   $$\frac{\partial c}{\partial (T-t)} = Sn(d_1)\frac{1}{2}\sigma\frac{1}{\sqrt{T-t}} + rKe^{-r(T-t)}N(d_2)$$

   $$\frac{\partial c}{\partial t} = -Sn(d_1)\frac{1}{2}\sigma\frac{1}{\sqrt{T-t}} - rKe^{-r(T-t)}N(d_2)$$

   Vega

   $$\frac{\partial c}{\partial \sigma} = S\sqrt{T-t}n(d_1)$$

   Rho (ρ)

   $$\frac{\partial c}{\partial r} = K(T-t)e^{-r(T-t)}N(d_2)$$

3. Ecrire une fonction calculant l'ensemble des dérivées ; Ecrire une fonction l'appliquant à l'exemple précédent.

Exercice 3 : Valeur approchée

Il est également intéressant de calculer la volatibilité impliquée par la valeur et le prix d'exercice c_0. Il faut donc trouver une solution à $c_0 = c(S, K, r, \sigma, T-t)$. Cependant, il n'y a pas de forme fermée, une solution est donc d'utiliser une recherche binomiale.

Il faut borner sigma entre un inférieur (fixé à $1.0e-5$). On cherche ensuite à fixer le supérieur juste au dessus de la valeur réelle, que nous fixons ici initialement à 0.3. Enfin, on effectue une recherche systematique entre les bornes en les réduisant au fur et à mesure de la recherche.

Implémenter cet algorithme en tenant compte de la faisabilité.

Astuces : La fonction doit prendre en paramètre S, K, r et t et doit estimer la valeur de σ en fonction d'un prix observé également transmis en paramètre. La fonction doit vérifier que le prix observé est

cohérent (sinon elle retourne une erreur ou 0 par exemple). Puis, en fixant une valeur de précision (ex. $1.0e-5$), le nombre d'itérations maximum (ex. 100), une valeur maximale (ex. $1e10$), une valeur de borne inférieur (ex. $1.0e-5$) et de borne supérieur (ex. 0.3) pour σ, la fonction doit :

- En partant du prix retourné par fonction calculant le prix d'un call d'une option de l'exercice 2, avec la valeur de la borne supérieure pour σ, et en doublant la valeur de la borne supérieure au fur et à mesure, trouver un prix supérieur au prix observé.
- Puis, sans dépasser le nombre d'itérations maximum, faire une recherche dichotomique de la valeur de sigma dans l'intervalle de valeurs formé de la borne inférieure de départ et de la borne supérieure, modifiée précédemment, jusqu'à ce que la différence entre le prix calculé et le prix observé soit inférieur à la précision.

Si on arrive à une différence de prix inférieure à la précision, la valeur de σ est retournée. En revanche, si on atteint le nombre maximum d'itérations, une erreur est retournée.

2 TP2 : Sur-définition ou surcharge de fonction et première classe

Exercice 4 : **Sur-définition ou surcharge de fonction**

Écrire les fonctions suivantes (sans utiliser la fonction `pow`) :

- `float puissance(int)` : fonction retournant le carré de l'entier passé en paramètre de la fonction.
- `float puissance(float)` : fonction retournant le carré du réel passé en paramètre de la fonction.
- `float pruissance(int i,int j)` : fonction retournant la valeur de `i` à la puissance `j`.
- `float pruissance(float f, int i)` : fonction retournant la valeur de `f` à la puissance `i`.

On suppose que les variables suivantes sont définies dans le programme principal :

```
int i, j;
float f;
char c;
```

Pour chacun des appels suivants :

1. Indiquer (sans tester sur machine) si l'appel est correct et, si oui, indiquer quelle fonction va être effectivement appelée et quelle conversion, s'il y a lieu, va être mise en place.
2. Vérifier le résultat de la première question sur machine.
   (a) `puissance(i) ;`
   (b) `puissance(f) ;`
   (c) `puissance(c) ;`
   (d) `puissance(i,j) ;`
   (e) `puissance(f,i) ;`

Exercice 5 : **Définition d'une classe `Fraction`**

1. Définir une classe `Fraction` ayant deux attributs de type entier privés `num` et `den` représentant respectivement le numérateur et le dénominateur d'une fraction.
2. Définir deux constructeurs pour cette classe :
   (a) Le premier constructeur, sans paramètre, construit une fraction dont le numérateur est 0 et le dénominateur est 1.
   (b) Le deuxième constructeur construit une fraction à partir du numérateur et du dénominateur transmis en paramètre.
3. Définir les méthodes suivantes :
   (a) `int Signe()` : retournant 1 si la fraction est positive ou nulle et -1 sinon.
   (b) `void Afficher()` : affichant la fraction sous la forme "num/den".

(c) `float Valeur()` : retournant la valeur décimale de la fraction.

(d) `void Simplifier()` : qui simplifie la fraction. Cette méthode nécessite l'écriture d'une fonction récursive `int pgcd (int a, int b)` qui calcule le pgcd de deux entiers `a` et `b`.

(e) `Fraction Inverse()` : retournant la fraction inverse.

4. Définir les fonction suivantes :

(a) `int compare(Fraction f, Fraction g)` : retournant 1 si `f`>`g`, 0 si les deux fractions sont égales et -1 si `f`<`g`.

(b) `Fraction somme(Fraction f, Fraction g)` : retournant la somme de deux fractions `f` et `g`.

(c) `Fraction difference(Fraction f, Fraction g)` : retournant la difference de deux fractions `f` et `g`.

(d) `Fraction multiplication(Fraction f, Fraction g)` : retournant le produit de deux fractions `f` et `g`.

(e) `Fraction division(Fraction f, Fraction g)` : retournant le quotient de deux fractions `f` et `g`.

3 TP3 : Manipulation de vecteurs

Pour ce TP, deux bibliothèques C++ sont nécessaires : `vector` et `cmath`.

`vector` qui définit un template de type vecteur (voir documentation à l'adresse `http ://cplusplus.com/reference/stl/vector/`) :

```
vector<type_element> nom_variable ; // déclaration d'une variable de type vecteur
contenant des valeurs de type type_element
void push_back ( const T x ) ; // méthode appliquée à un vecteur qui insére l'élément
x (de type type_element) à la fin du vecteur
reference operator[] ( size_type n ) ; // surcharge de l'opérateur crochets permettant
d'utiliser les crochets pour accèder aux éléments du vecteur
```

Voir exemple de manipulation de vector à la fin de ce document

`cmath` qui contient en particulier les fonctions suivantes :

```
double exp ( double x ) ;
double pow ( double x, double y ) ;
double sqrt ( double x ) ;
```

voir documentation à l'adresse `http ://www.cplusplus.com/reference/clibrary/cmath/`)

Exercice 6 : **Modèle Binomial à n périodes**

Le modèle binomial est particulièrement utilisé dans le cadre d'options américaines ou exotiques. Pour des raisons de simplicité, nous étudions le cas appliqué aux options européennes.

L'évolution du prix dans le modèle de Black-Scholes peut être approchée par un arbre binomial en utilisant :

$$S_{0+t} = \begin{cases} u \times S_0, & q \\ d \times S_0, & 1-q \end{cases}$$

La probabilité q est calculée telle que $q = \frac{e^r - d}{u - d}$, et u et d sont les données (évolution haute et basse) obtenues par $u = e^{\sigma\sqrt{t}}$ et $d = \frac{1}{u}$.

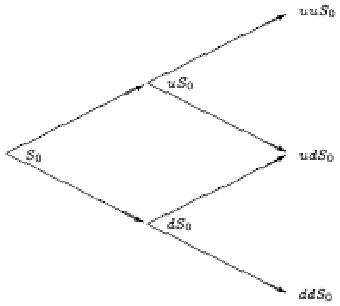

FIG. 1 – Arbre binomial pour deux périodes.

Algorithme

Il s'agit de construire l'ensemble des feuilles de l'arbre, puis de remonter jusqu'à la racine pour trouver la valeur de l'option.

Notons S_n^m le m-ième noeud de l'arbre à l'instant n, avec S_0 le prix actuel :

$$S_n^m = (1+d)^{n-m}(1+u)^m S_0 \tag{1}$$

Notons C_n^m le prix d'une option européenne à l'instant n étant donné que l'action vaut S_n^m. Pour le calcul des feuilles, on a $C_n^m = max(0, S_n^m - X)$ avec X le prix du strike.

On peut ensuite remonter l'arbre par la formule d'itération suivante :

$$C_n^m = e^{-r}(qC_{n+1}^{m+1} + (1-q)C_{n+1}^m) \tag{2}$$

avec r le taux d'intérêt par période. La racine de l'arbre donne l'évaluation de l'option.

A faire

Implémenter une fonction qui calcule le prix d'une option Call en utilisant le modèle binomial à périodes multiples, le nombre de périodes étant un paramètre.

Astuce

Vous pouvez utiliser un vecteur de vecteurs de double (qui se déclare par `vector < vector<double> > nom_variable` pour implémenter l'arbre. Le premier élément du vecteur de vecteurs contiendra un vecteur contenant une seule valeur (la racine de l'arbre), le deuxième élément du vecteur de vecteurs contiendra un vecteur à deux valeurs, l'élément suivant un vecteur contenant trois valeurs etc. Vous pouvez également utiliser un tableau de tableaux de `double`.

Exemple de manipulation des vecteurs

```
int main ()
{
  ...

  // Déclaration d'un vecteur de doubles de taille N
  vector<double> niveau_arbre (N);

  // Déclaration d'un vecteur de vecteur de doubles
  vector< vector<double> > arbre;

  ...
```

```
// Accès au ième élément du vecteur
niveau-arbre[i]=x; //niveau-arbre[i] est de type double

...

// Ajout du vecteur niveau-arbre dans le vecteur de vecteurs nommé arbre
arbre.push_back(niveau_arbre);

...

//Accès au ième élément du jème vecteur de arbre
arbre[j][i]=y; //arbre[j] est de type vector<double> et arbre[j][i] est de type double

...


}
```

Attention : les indices des vecteurs commencent à zéro.

4 TP4 : Héritage

Pour ce TP, deux bibliothèques C++ sont nécessaires : `vector` et `cmath`.

`vector` qui définit un template de type vecteur (voir documentation à l'adresse `http ://cplusplus.com/reference/stl/vector/`) :

```
vector<type_element> nom_variable ; // déclaration d'une variable de type vecteur
contenant des valeurs de type type_element
void push_back ( const T x ) ; // méthode appliquée à un vecteur qui insére l'élément
x (de type type_element) à la fin du vecteur
reference operator[] ( size_type n ) ; // surcharge de l'opérateur crochets permettant
d'utiliser les crochets pour accèder aux éléments du vecteur
```

Voir exemple de manipulation de vector de l'exercice précédent

`cmath` qui contient en particulier les fonctions suivantes :

```
double exp ( double x ) ;
double pow ( double x, double y ) ;
double sqrt ( double x ) ;
```

voir documentation à l'adresse `http ://www.cplusplus.com/reference/clibrary/cmath/`)

Exercice 7 : Calculer le facteur d'actualisation, le taux au comptant et le taux à termes d'un flux de trésorerie

L'objectif de ce TP est d'implémenter une structure à terme des taux d'intérêt (*term structure of interest rates*), i.e. la relation entre la durée d'une période d'investissement et le taux d'intérêt. Une structure à terme peut être spécifiée par un facteur d'actualisation (*discount factor*), un taux au comptant (*spot rate*) ou un taux à termes ou anticipé (*forward rate*). Le facteur d'actualisation associé à une date future t est le prix qu'il faudrait payer aujourd'hui pour acheter un euro perçu à cette date future. Le facteur d'actualisation correspond donc à la quantité d'argent qu'il serait nécessaire de placer aujourd'hui pour obtenir 1 euro à cet horizon (capitalisation). Ce facteur, noté d_t, se calcule à partir du taux au comptant r_t :

$$d_t = e^{-r_t t}$$

Le taux au comptant peut également se calculer à partir du facteur d'actualisation :

$$r_t = \frac{-ln(d_t)}{t}$$

Le taux à terme t_2 d'un emprunt à la date t_1 j se calcule, quant à lui, à partir de d_t ou de r_t :

$$f_{t_1,t_2} = \frac{ln(\frac{d_{t1}}{d_{t2}})}{t_2 - t_1} = r_{t2}\frac{t_2}{t_2 - t_1} - r_{t1}\frac{t_1}{t_2 - t_1}$$

1. Ecrire les fonctions C++ permettant de calculer d_t, r_t et f_{t_1,t_2}.
2. Définir une classe `StructureATerme` comportant trois attributs t_1, t_2 et r (taux au comptant), un constructeur à trois attributs et trois méthodes appelant les fonctions définies précédemment pour calculer d_t, r_t et f_{t_1,t_2}. Tester votre classe avec $t_1 = 1$, $t_2 = 2$ et $r = 5\%$.
3. Ecrire une fonction permettant de calculer le taux au comptant à une date T par interpolation linéaire à partir de deux vecteurs, un vecteur de dates et un vecteur de taux au comptant. Vous testerez votre fonction en prenant les valeurs suivantes :

Date	**Taux au comptant**
0.1	0.1
0.5	0.2
1	0.3
5	0.4
10	0.5

   Tester votre fonction en calculant le le taux au comptant à la date $T = 3$.
4. Définir une classe `StructureATermeInterpolee` comportant quatre attributs, t_1, t_2, un vecteur de dates et un vecteur de taux au comptant, un constructeur prenant en paramètre quatre attributs, et trois méthodes appelant les fonctions définies précédemment pour calculer d_t, r_t et f_{t_1,t_2}. Tester votre classe avec $t_1 = 1$, $t_2 = 2$ et le tableau ci-dessous :

Date	**Taux au comptant**
0.1	0.05
1	0.07
5	0.08

5 TP5 : Création de DLL et appel de la DLL sous Excel (suite)

Pour ce TP, la bibliothèques C++ `cmath` est nécessaire. Elle contient en particulier les fonctions suivantes :

```
double exp ( double x ) ;
double pow ( double x, double y ) ;
double sqrt ( double x ) ;
```

voir documentation à l'adresse `http ://www.cplusplus.com/reference/clibrary/cmath/`)

Exercice 8 : valuation des warrants

Les warrants sont des valeurs mobilières de type optionnel qui offrent le droit d'acheter (call) ou de vendre (put) à un cours déterminé à l'avance (prix d'exercice ou strike) un actif jusqu'à une date prévue dès l'origine. Une nouvelle action est alors émise.

Soient K le prix de l'option, n le nombre de parts et m le nombre de warrants émis. A est la valeur des actifs de la firme, et C_{BS} la valeur d'un call par la méthode de Black-Scholes[2].

[2]Fonction que vous avez déjà développé en C++ lors du TP1.

Le prix du warrant est ainsi :

$$W_t = \frac{n}{n+m} C_{BS}\left(\frac{A}{n}, K, \sigma, r, (T-t)\right)$$

Nous cherchons à effectuer la valuation des warrants en observant la valeur actuelle (*stock value*).
La valeur de l'actif est, en fonction du sous-jacent :

$$A_t = nS_t + mW_t$$

La valeur du warrant devient alors :

$$W_t = \frac{n}{n+m} C_{BS}\left(S_t + \frac{m}{n} W_t, K, \sigma, r, (T-t)\right)$$

W_t étant des deux côtés de l'équation, nous recherchons une solution numérique. Pour cela, nous utilisons la méthode Newton-Rhapson :

$$g(W_t) = W_t - \frac{n}{n+m} C_{BS}\left(S_t + \frac{m}{n} W_t, K, \sigma, r, (T-t)\right)$$

En commençant par une valeur initiale de la valeur de W_{t_0}, l'itération est la suivante :

$$W_t^i = W_t^{i-1} - \frac{g(W_t^{i-1})}{g'(W_t^{i-1})},$$

L'itération continue tant que l'approximation $g(W_t^{i-1})$ est au dessus d'un certain seuil ϵ.
Nous précisons :

$$g'(W_t) = 1 - \frac{m}{m+n} N(d_1)$$

avec

$$d_1 = \frac{\ln\left(\frac{S_t + \frac{m}{n} W_t}{K}\right) + (r + \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}}$$

.

Une valeur initiale pour W_{t_0} est le prix actuel $\frac{n}{n+m} C_{BS}(S, K, \sigma, r, (T-t))$.

1. Coder la fonction de valuation des warrants par la méthode de Newton-Rhapson en C++ dans une DLL C++. Les deux fonctions appelables de la DLL sont (1) la fonction de calcul de Black-Scholes et (2) la fonction de valuation de warrants. Ces deux fonctions seront appelées dans un module VBA d'un fichier Excel.
2. Créer un fichier Excel contenant les valeurs suivantes :

	A	B	C	D	E	F	G	...
1	S	K	r	σ	t	m	n	...
2	48	40	0.08	0.30	0.5	1000	10000	...
3								
4						resultat	—–	

3. Créer une macro utilisant la fonction de valuation de warrants de la DLL sur les valeurs du fichier Excel, et les affichant en cellule G4.

Exercice 9 : **Modèle Binomial**
Le modèle binomial est particulièrement utilisé dans le cadre d'options américaines ou exotiques. Pour des raisons de simplicité, nous étudions le cas appliqué aux options européennesà une période.
L'évolution du prix dans le modèle de Black-Scholes peut être approchée par un modèle binomial en utilisant les formules suivantes :

$$S_u = u \times S_0$$

$$S_d = d \times S_0$$

$$C_u = max(0, S_u - K)$$

$$C_d = max(0, S_d - K)$$

$$C_0 = e^{-r}(qC_u + (1-q)C_d)$$

avec S_0 le prix actuel, u l'évolution haute, d l'évolution basse, r le taux d'intérêt et K le prix du strike. La probabilité q est calculée par la formule $q = \frac{e^r - d}{u - d}$.

1. Coder une DLL C++ permettant de calculer ces formules.
2. Créer un fichier Excel contenant les valeurs suivantes :

	A	B	C	D	E	F	G	...
1						r=		...
2	Nom	Valeur Actuelle	Option	Eval Haute	Eval Basse			...
3	Action1	154,02	200,22					...
4	Action2	108,23	218,7					...
5	Action3	161,31	209,7					...
6	Action4	103,71	134,82					...
7	Action5	125,29	162,87					...
8	Action6	53,08	69,01					...

3. Créer une fonction VBA qui affecte des valeurs aléatoire comprises entre 0,3 et 0,9 dans la colonne `Eval Basse` et des valeurs aléatoire comprises entre 1,1 et 1,5 dans la colonne `Eval Haute` (cf. exemple dans les transparents de cours).
4. Appeler les focntions de votre DLL pour calculer le prix des options selon les formules ci-dessus et en stockant ce prix dans la colonne F, le taux d'intérêt étant saisi dans la cellule G1.