CentraleSupélec

Travaux Pratiques
Modélisation de Risques Financiers

Auteurs

Bataillou Almagro Marc — Cobas Aranguren Luís

—

2016

Table des matières

1 Formule de Black et Scholes **4**

2 Grecques d'un risk-reversal **8**

3 Arbre Binomial **12**
3.1 Prix du CALL ou du PUT avec 20 pas 12
3.2 Analyse de la variation du prix de l'option en fonction du nombre de pas 13
3.3 Frontiere d'exercice d'une option americaine 16

4 Calcul du delta **18**

5 Monte-Carlo **20**
5.1 Analyse du CALL et du PUT 20
5.2 Analyse pour le delta 22

6 Option barrière **23**
6.1 Méthode de Monte-Carlo 24
6.2 Modèle Binomial 26

7 Codes **28**

Table des figures

1 Variation du prix du CALL en fonction du sous-jacent S_0 5
2 Variation du prix du CALL en fonction de la volatilité σ 6
3 Variation du prix du CALL en fonction date d'échéance T 7
4 Variation du δ d'un RISK-REVERSAL en fonction de S_0 9
5 Variation du γ d'un RISK-REVERSAL en fonction de S_0 9
6 Variation du ν d'un RISK-REVERSAL en fonction de S_0 10
7 Variation du θ d'un RISK-REVERSAL en fonction de S_0 10
8 Variation du prix du CALL en fonction date d'échéance T 12
9 Variation du prix du CALL européen en fonction du nombre de pas 13
10 Variation du prix du CALL européen en fonction du nombre de pas 14
11 Variation du prix du PUT européen en fonction du nombre de pas 14
12 Variation du prix du PUT américain en fonction du nombre de pas 15
13 Variation de l'ensemble des options en fonction du nombre de pas 16
14 Frontiere d'exercice d'un put pour un nombre de pas $n = 200$ . . . 17
15 Frontiere d'exercice d'un put pour un nombre de pas $n = 2000$ . . 17
16 Variation du δ d'un CALL européen approché par la méthode de différences finies en fonction du nombre de pas 18
17 Variation du δ d'un PUT européen approché par la méthode de différences finies en fonction du nombre de pas 19
18 Variation du prix du CALL européen approché par la méthode de Monte-Carlo en fonction du nombre de tirages 21
19 Variation du prix du PUT européen approché par la méthode de Monte-Carlo en fonction du nombre de tirages 21
20 Variation du δ d'un CALL européen approché par la méthode de Monte-Carlo en fonction du nombre de tirages 22
21 Variation du δ d'un PUT européen approché par la méthode de Monte-Carlo en fonction du nombre de tirages 22
22 Payoff d'une option barrière sur un PUT européen en fonction du cours du sous-jacent 23
23 Payoff d'une option barrière sur un CALL européen en fonction du cours du sous-jacent 24
24 Variation d'une option européenne avec option barrière approché par la méthode de Monte-Carlo en fonction du nombre de tirages . 24
25 Variation d'une option européenne avec option barrière approché par la méthode de Monte-Carlo en fonction du niveau de la barrière 25
26 Variation d'une option européenne avec option barrière approché par le modèle binomial en fonction du pas 26
27 Variation d'une option européenne avec option barrière approché par le modèle binomial en fonction du niveau de la barrière 27

Codes

1 Calcul du CALL et du PUT via la méthode de Black and Scholes. Matlab. 28
2 Calcul des grecques du CALL dans le modèle de Black et Scholes . 29
3 Fonction de calcul des valeurs nécessaires au modèle binomial . . 30
4 Simulation du payoff aleatoire dans le modèle binomial 30
5 Algorithme de calcul du prix d'une option via le modèle binomial 31
6 Calcul de la frontiere d'exercice d'une option americaine (temps,S_t) 32
7 Algorithme de calcul du delta d'une option via le modèle binomial 33
8 Algorithme de calcul du prix d'une option européenne via la simulation de Monte-Carlo 33
9 Algorithme de calcul du delta d'une option européenne via la simulation de Monte-Carlo 34
10 Algorithme de calcul du prix d'une option barrière via la simulation de Monte-Carlo 35
11 Algorithme de calcul du prix d'une option barrière via le modèle binomial 36

1 Formule de Black et Scholes

Les fonctions permettant de calculer les différents influences et valeurs de cette section sont Listing 1.

Nous présentons dans les Figure 1, Figure 2, et Figure 3, l'évolution du prix d'un CALL évalué selon la formule de Black et Scholes.

$$C_t = S_t N(d_1) - K e^{-r(T-t)} N(d_2) \tag{1}$$

où

$$\begin{aligned} N(d) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{d} e^{\frac{-q^2}{2}} dq \\ d_1 &= \frac{1}{\sigma\sqrt{T-t}} (ln(\frac{S}{K}) + (r + \frac{\sigma^2}{2})(T-t)) \\ d_2 &= d_1 - \sigma\sqrt{T-t} \end{aligned}$$

On pourra se référer au document BLACK_SCHOLES.M pour analyser l'implémentation de la formule sous MATLAB. Pour simuler ce prix, on va prendre les paramètres suivants : le prix du sous-jacent $S_0 = 75$, le Strike $K = 75$, l'échéance $T = 1$, la volatilité $\sigma = 0.17$ et le taux d-actialisation $r = 0.01$. Avec ces paramètres, on obtient le prix d'un call européen $P_{B-S} = 5.437$ en applicant le formule de Black et Scholes.

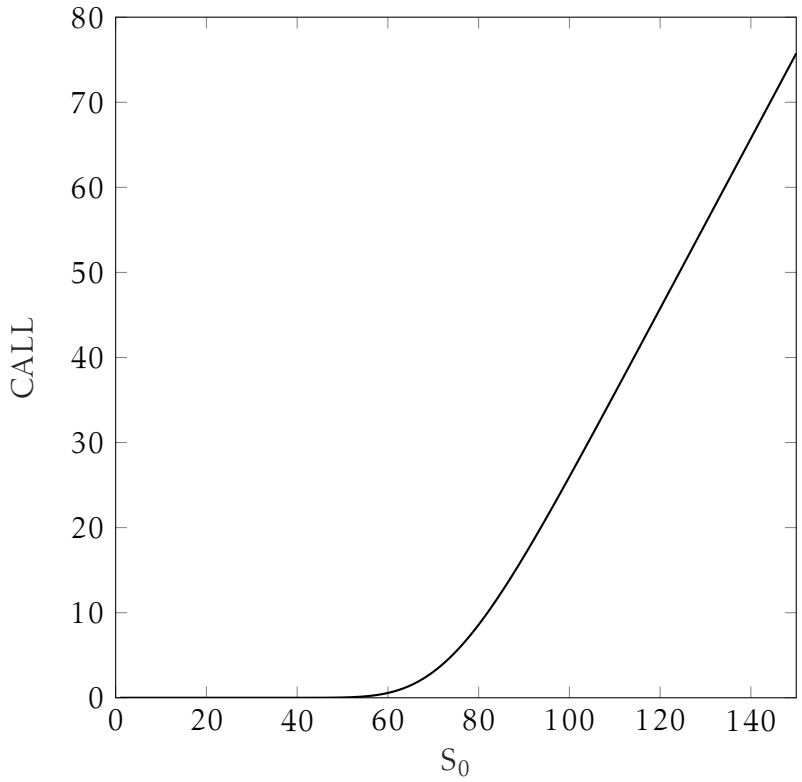

FIGURE 1 – Variation du prix du CALL en fonction du sous-jacent S_0

On voit une fonction convexe croissante en fonction de l'evolution de S_0. On remarque par ailleurs que la courbe tend vers la droite $(S_0 - K)_+$.

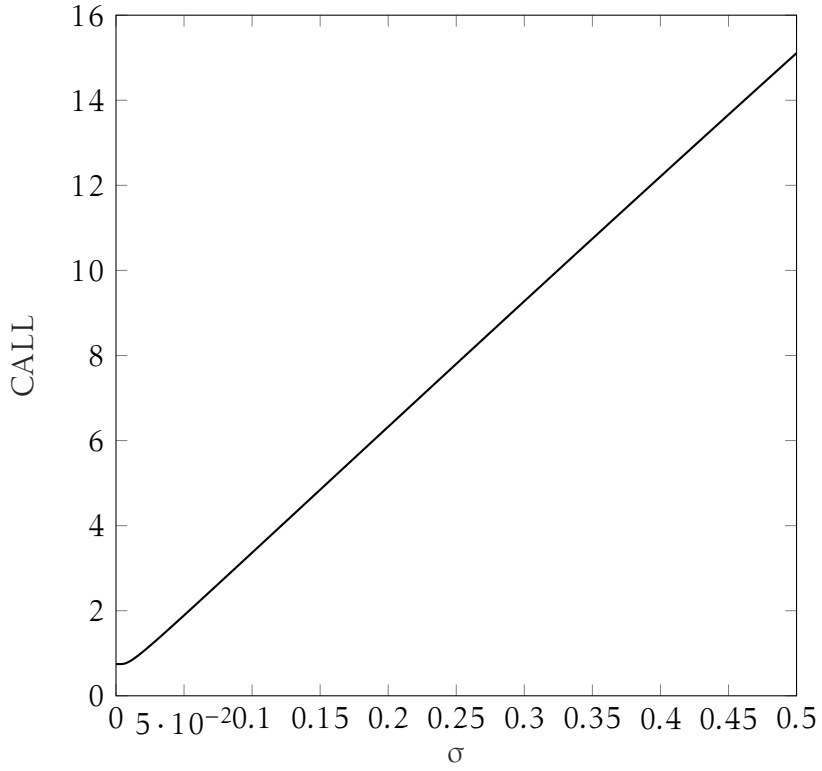

FIGURE 2 – Variation du prix du CALL en fonction de la volatilité σ

En faisant varier la volatilité σ, on obtient un graphe qui semble linéaire à partir d'une certaine valeur. La fonction est croissante ce qui est lié au fait que plus la volitilité est élévée plus les gains potentiels sont importants (il ne faut pas oublier que les investisseurs sont *risk-averse*). Néanmoins, si la volatilité du prix du sous jacente est nulle, le détenteur de l'option n'a aucune perspective de gains et revend son option à un prix "nul" (ce qui concorde avec le graphique).

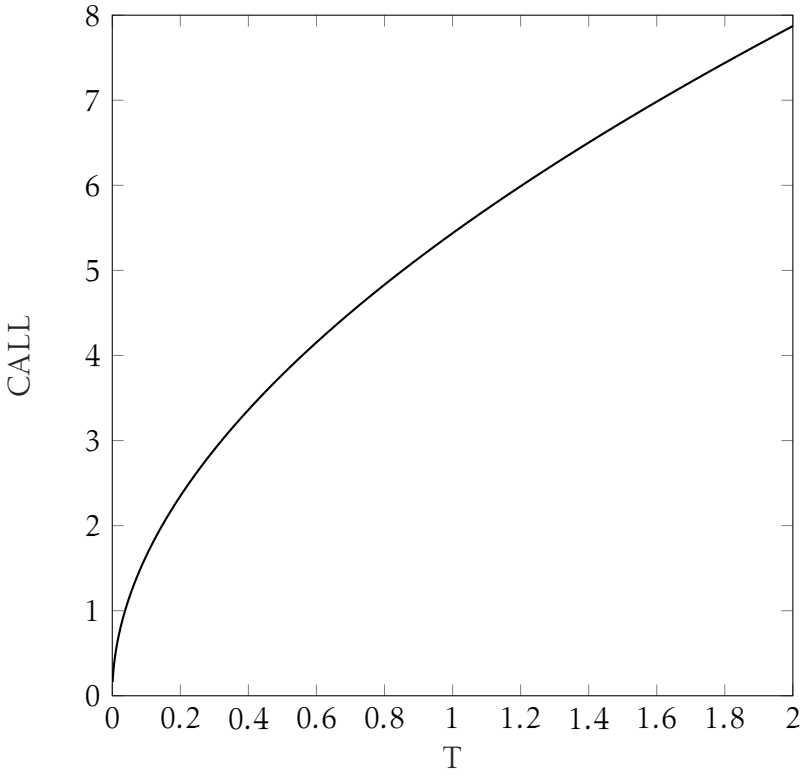

Figure 3 – Variation du prix du CALL en fonction date d'échéance T

On constante une fonction concave et croissante. C'est logique que le prix soit croissant avec la date d'échéance T. Si T est plus loin, plus les perspectives de gains sont élevées. La concavité est liée au fait que si la date d'écheance est grande, avoir un jour en plus/en moins ne change rien pour de grandes valeurs de T.

2 Grecques d'un risk-reversal

Les différentes fonctions qui permettent de calculer les grecques sont Listing 6 et sont implémentées dans le fichier GRECQUES.M.

Un risk-reversal est la différence entre un put de strike K_1 et un call de strike $K_2 > K_1$. En plus, en mathématiques financières les principales grecques sont :

1. Sensibilité de la prime au sous-jacent : **DELTA**
   Pour un call :
   $$\Delta_c = \frac{\partial C_t}{\partial S} = N(d_1) > 0$$
   Pour un put :
   $$\Delta_p = N(d_1) - 1 < 0$$
2. Sensibilité du delta au sous-jacent : **GAMMA**
   Pour un call-put :
   $$\Gamma_c = \Gamma_p = \frac{\partial^2 C_t}{\partial S^2} = \frac{n(d_1)}{S\sigma\sqrt{T-t}} > 0$$
3. Sensibilité de la prime à la volatilité : **VEGA**
   Pour un call :
   $$V_c = \frac{\partial C_t}{\partial \sigma} = n(d_1)S\sqrt{T-r} > 0$$
   Pour un put :
   $$V_c = e^{-r(T-t)}n(d_1)S\sqrt{T-r} > 0$$
4. Sensibilité de la prime au taux d'intérêt : **THÊTA**
   Pour un call :
   $$\Theta_c = \frac{\partial C_t}{\partial T} = n(d_1)\frac{S\sigma}{2\sqrt{T-t}} + rKe^{-r(T-t)}N(d2) > 0$$
   Pour un put :
   $$\Theta_p = n(d_1)\frac{S\sigma}{2\sqrt{T-t}} + rKe^{-r(T-t)}(N(d2) - 1)$$

Par linéarité nous obtenons les grecques d'un risk-reversal :

$$\Delta_{RR} = \Delta_p(K_1) - \Delta_c(K_2) \quad (2)$$
$$\Gamma_{RR} = \Gamma_p(K_1) - \Gamma_c(K_2) \quad (3)$$
$$V_{RR} = V_p(K_1) - V_c(K_2) \quad (4)$$
$$\Theta_{RR} = \Theta_p(K_1) - \Theta_c(K_2) \quad (5)$$

Dans notre cas on va utiliser les valeurs $K_1 = 75$ et $K_2 = 80$.

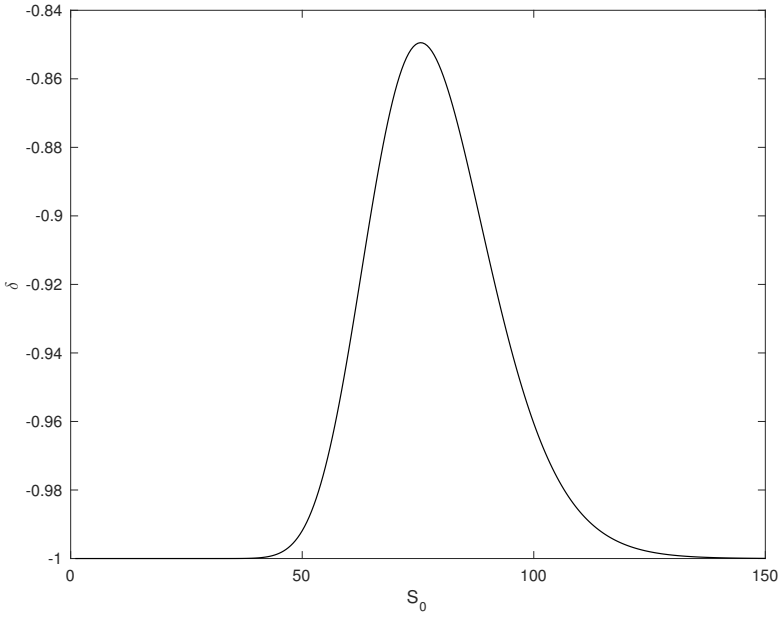

FIGURE 4 – Variation du δ d'un RISK-REVERSAL en fonction de S_0

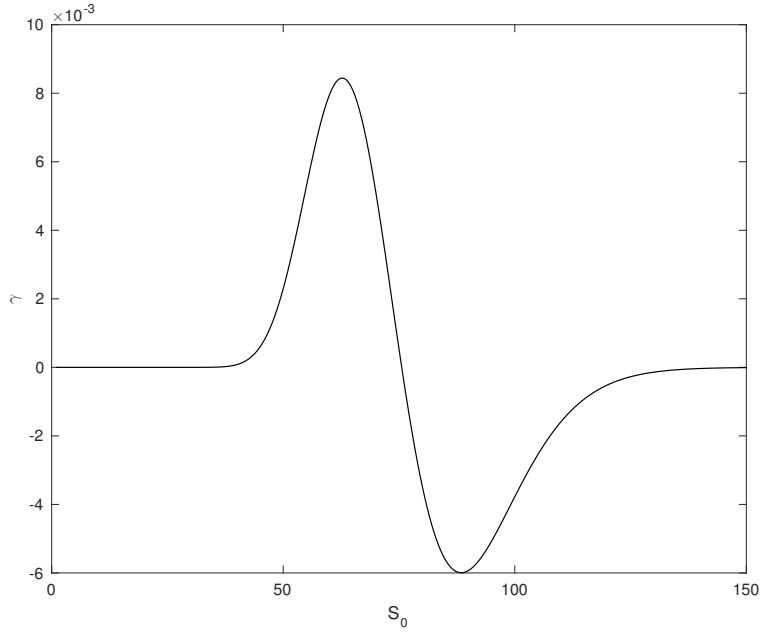

FIGURE 5 – Variation du γ d'un RISK-REVERSAL en fonction de S_0

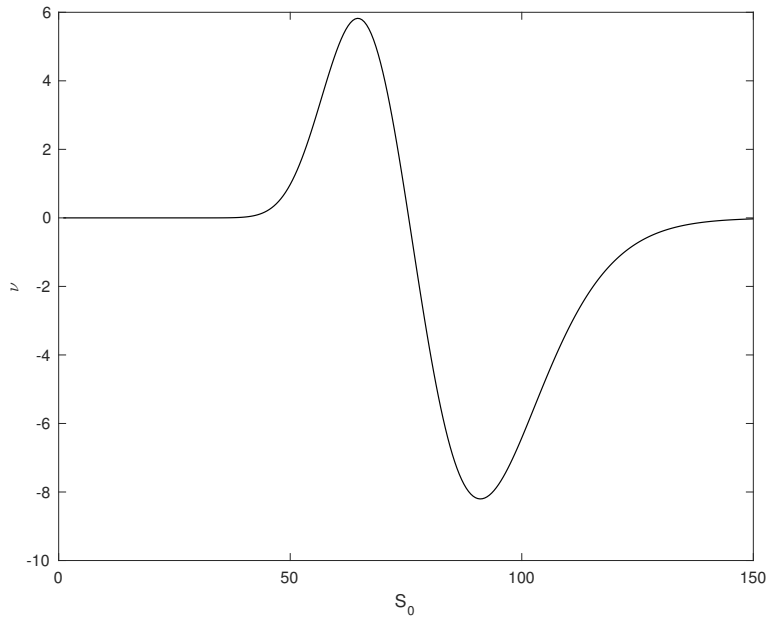

FIGURE 6 – Variation du ν d'un RISK-REVERSAL en fonction de S_0

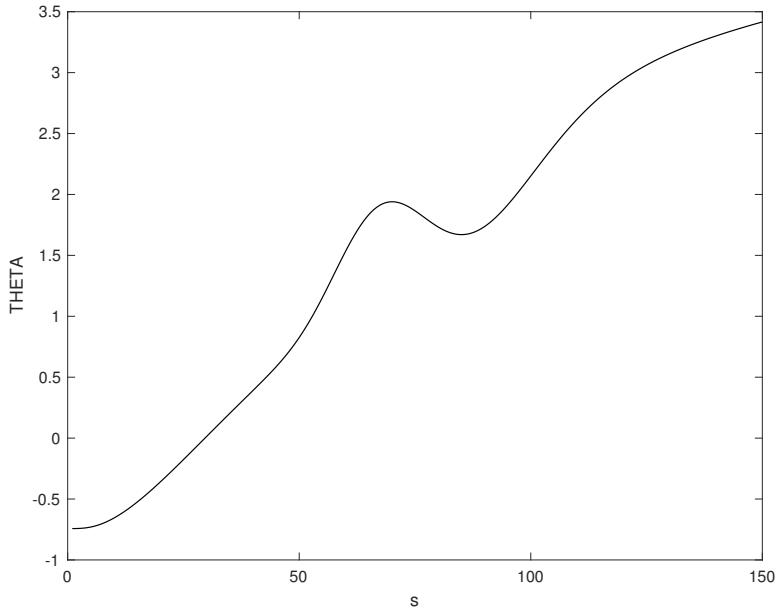

FIGURE 7 – Variation du θ d'un RISK-REVERSAL en fonction de S_0

On remarque dans l'ensemble des figures un point important autour (S_0 = 75 – 85). Dans la Figure 4 on trouve le maximum du delta, ainsi que des points d'inflexion sur les figures : Figure 5, Figure 6, Figure 7.

Les résultats seraient à comparer avec la valeur d'un portefeuille contenant

un risk-reversal, on pourrait donc faire coincider ses variations avec les signes des dérivées, et donc des grecques obtenues.

3 Arbre Binomial

On analyse dans cette section, le prix du CALL ou du PUT à l'aide du modèle binomial. La Figure 8 nous donne un reprèsentation simplifiée du modèle, qui aide à comprendre l'implémentation des algorithmes réalisés. La fonction permettant de calculer ces valeurs est Listing 5, elle est implémentée dans le fichier ARBRE.PY.

Note : Dans le fichier code ARBRE.PY, *on a laissé plusieurs implémentations pour réaliser de mêmes calculs. La raison de ce choix est que certaines implémentations, n'étaient pas assez efficientes et causaient un* ***Stackoverflow****. On les a marqué comme* ***"optionelles"****.*

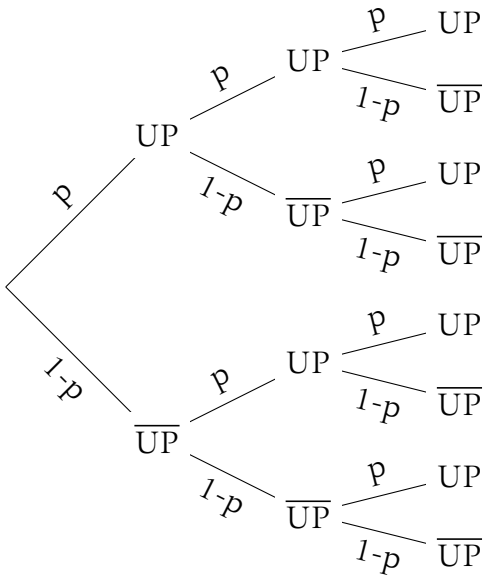

FIGURE 8 – Variation du prix du CALL en fonction date d'échéance T

3.1 Prix du CALL ou du PUT avec 20 pas

Option	Prix de l'option
CALL européen	5.37
CALL américain	5.37
PUT européen	4.63
PUT américain	4.70

TABLE 1 – Prix des différentes options calculées par récurrence avec le modèle binomial

3.2 Analyse de la variation du prix de l'option en fonction du nombre de pas

On observe sur la Figure 9 et Figure 11, que le prix des options européennes tendent vers le prix obtenus avec le modèle de Black et Scholes. Ce résultat était prévisible, de part la **construction** du modèle de Black et Scholes :

$$\lim_{pas \to \infty} \text{PRIX(CALL)}_{\text{BINOM}} = \text{PRIX(CALL)}_{\text{BLACKETSCHOLES}}$$

Pour ce qui est des options américaines on observe sur la Figure 10, et Figure 12 que $\text{CALL}_{amer} = \text{CALL}_{euro}$ et $\text{PUT}_{amer} \geq \text{PUT}_{euro}$. Ce qui est cohérent avec le **principe de non-domination** (AOA). En effet on a d'après la parité CALL, PUT et le **principe de non-domination** (AOA) :

$$\text{CALL}_T(T,K) - \text{PUT}_T(T,K) = S_T - K$$
$$\Leftrightarrow \text{CALL}_t(T,K) - \text{PUT}_t(T,K) = S_t - KB_t(T) \text{ (AOA)}$$

On obtiens donc les bornes suivantes :

$$(S_t - K)^+ \leq \text{CALL}_t(T,K) \leq S_t \qquad (KB_t(T) - S_t)^+ \leq \text{PUT}_t(T,K) \leq KB_t(T)$$

De plus il est évident que $\text{CALL}_{amer} \geq \text{CALL}_{euro}$ et $\text{PUT}_{amer} \geq \text{PUT}_{euro}$ puisqu'elles peuvent être éxecutées à tout moment, ce qui explique Figure 12. Cependant d'après les bornes antérieures on a :

$$\text{CALL}_{amer} \geq \text{CALL}_{euro} > (S_t - K)^+ \quad \forall t < T$$

Il n'est donc jamais optimal d'exercer le call américain avant échéance (s'il n'y a pas de dividendes), d'où l'égalité observée sur Figure 9.

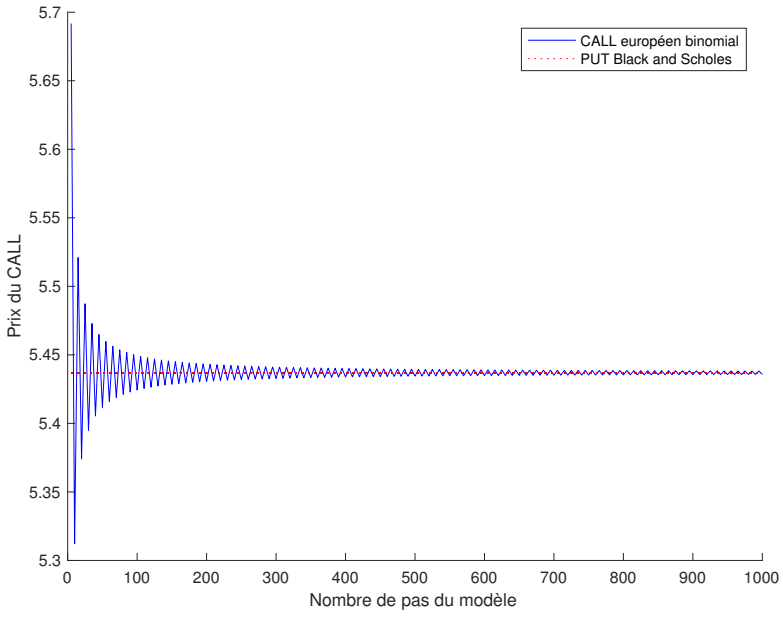

FIGURE 9 – Variation du prix du CALL européen en fonction du nombre de pas

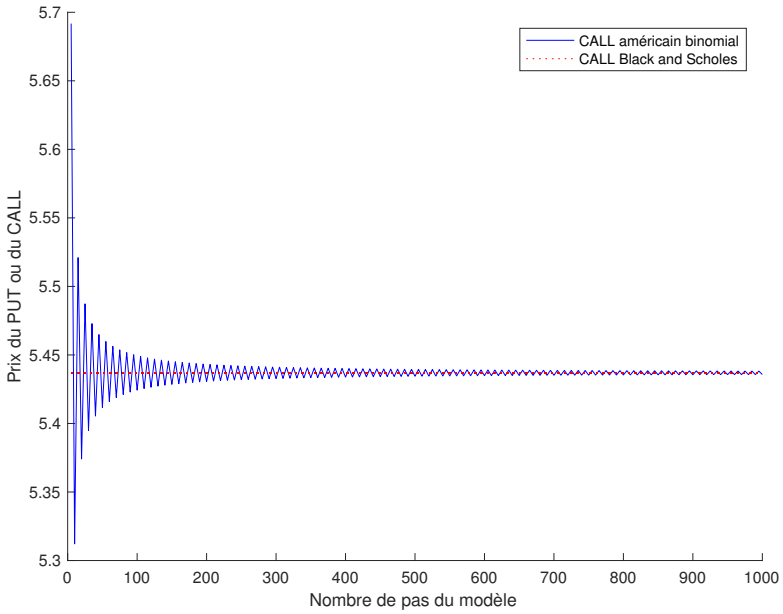

FIGURE 10 – Variation du prix du CALL européen en fonction du nombre de pas

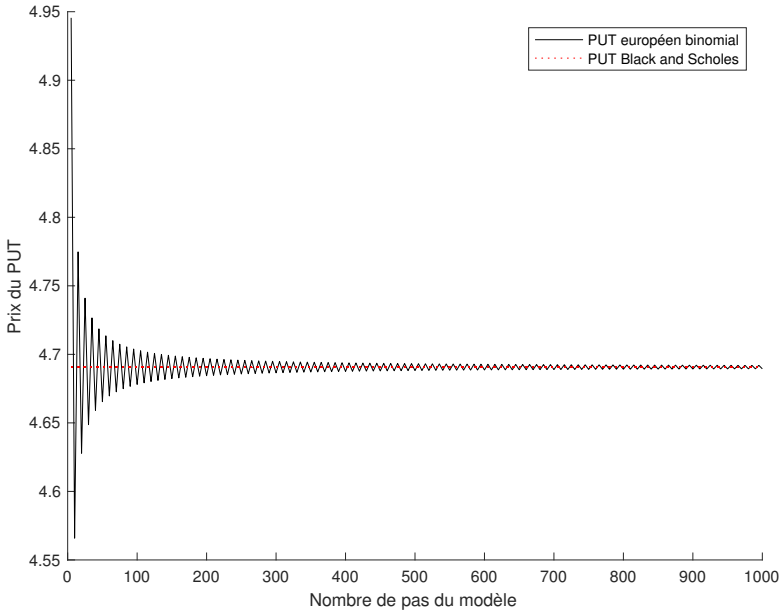

FIGURE 11 – Variation du prix du PUT européen en fonction du nombre de pas

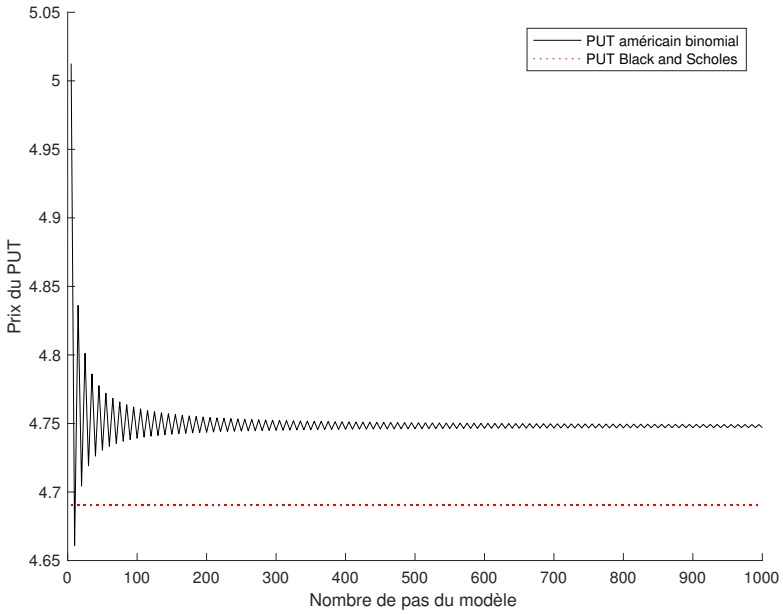

FIGURE 12 – Variation du prix du PUT américain en fonction du nombre de pas

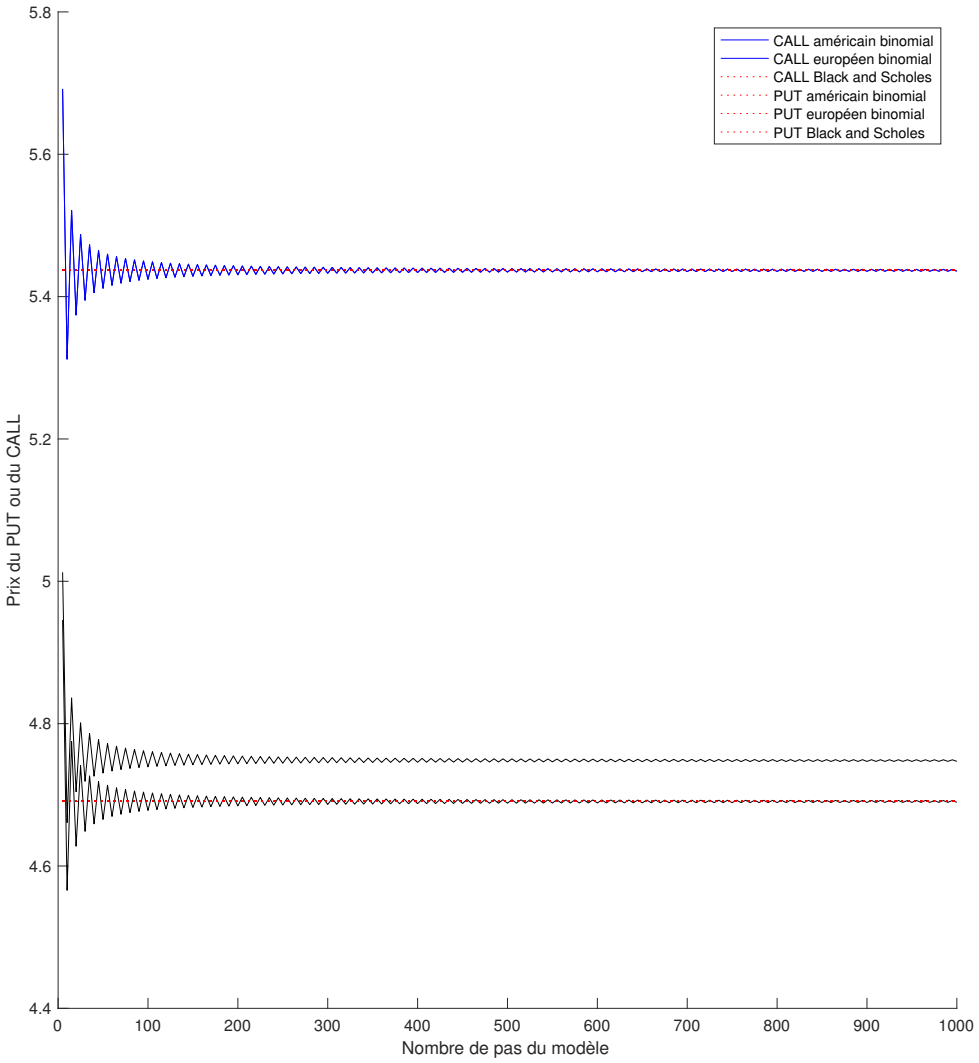

FIGURE 13 – Variation de l'ensemble des options en fonction du nombre de pas

On observe donc avec les graphiques ce que l'on a demontré au début de la section. Le prix d'un put américain est plus éleve que celui d'un put européen. Ce qui est lié au degré de liberté supplémentaire : elles peuvent-être exercées à n'importe quel moment. Le CALL lui est égal dans les 2 cas comme démontré antérieurement.

3.3 Frontiere d'exercice d'une option americaine

On calcule ensuite à l'aide de la fonction Listing 6 du code ARBRE.PY, la frontiere d'exercice d'un PUT. Il est inutile de calculer celles d'un CALL car comme

on l'a vu avant il n'est jamais exercé avant la date d'échéance, on a cependant implémenté une option pour le vérifier (elle donne évidemment []).

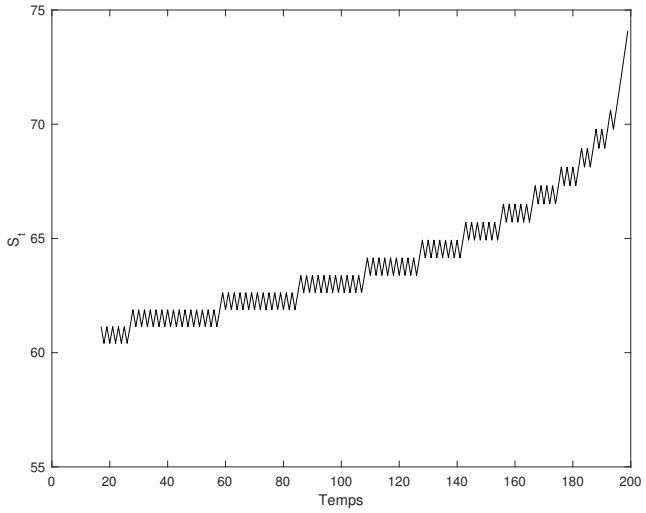

FIGURE 14 – Frontiere d'exercice d'un put pour un nombre de pas $n = 200$

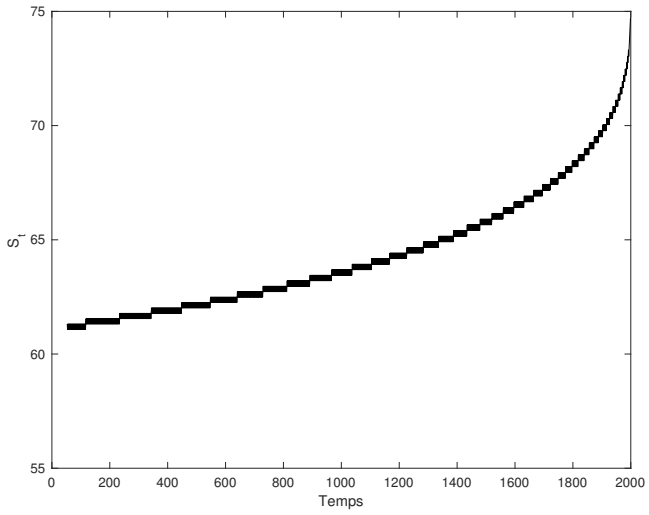

FIGURE 15 – Frontiere d'exercice d'un put pour un nombre de pas $n = 2000$

4 Calcul du delta

On sait par définition que le delta d'un call européen vaut

$$\Delta_c = \frac{\partial C}{\partial S}$$

Cette équation peut être approché par la méthode des différences finies centrées :

$$\Delta_{DF} = \frac{C(S_0 + \epsilon) - C(S_0 - \epsilon)}{2\epsilon}$$

En utilisant la méthode de Black et Scholes on obtient un delta d'un call européen avec le paramètres suivants : $S_0 = 75$, le strike K=75, l'écheance T = 1, la volatilité = 0.17 et le taux d'actualisation r = 0.01 qui vaut Δ_{B-S}=0.5572.

Pour analyser l'influence de la profondeur de l'arbre, il faut donc calculer C(So+ϵ) et C(So-ϵ) à l'aide de la méthode de la Section 3 afin d'obtenir par différences finies (avec ϵ=0.001), le résultat voulu. On obtient les Figure 16, Figure 17. Cette formule a été implémentée dans la fonction Listing 7, et on peut la tester dans le fichier DELTA.PY.

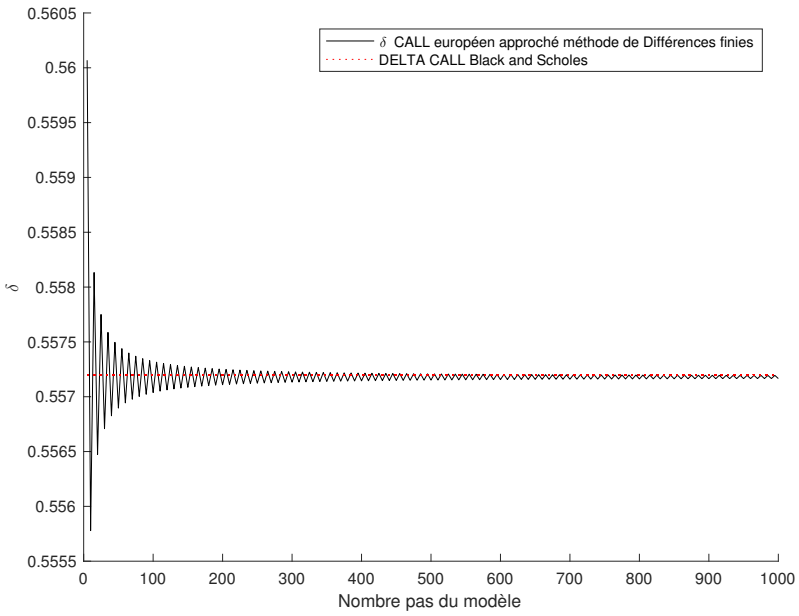

FIGURE 16 – Variation du δ d'un CALL européen approché par la méthode de différences finies en fonction du nombre de pas

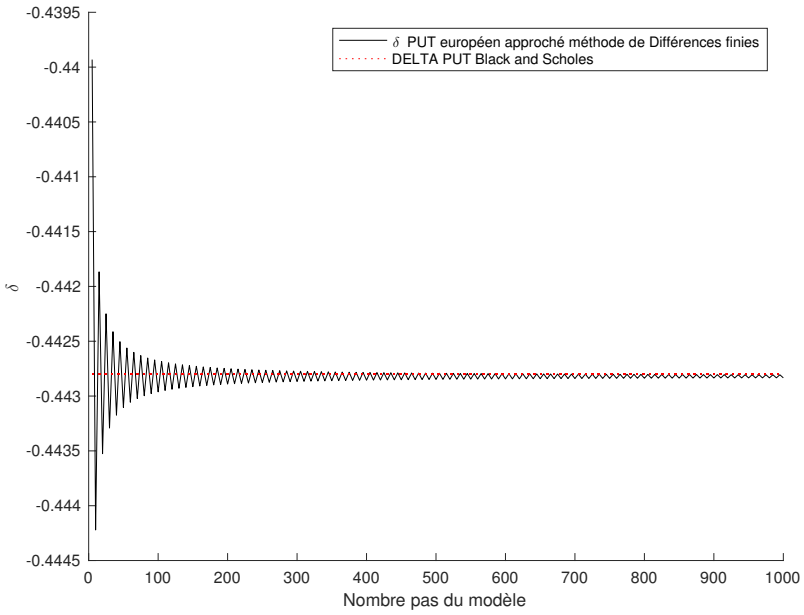

FIGURE 17 – Variation du δ d'un PUT européen approché par la méthode de différences finies en fonction du nombre de pas

Il est clair que plus le nombre de pas augmente, plus l'amplitude de variation autour de δ diminue. De plus la vitesse de convergence est clairement rapide.

5 Monte-Carlo

Soit $(Z_i) =_{i=1,....,N}$ une suite de variables i.i.d telles que Z_i suit une loi normale standard $\mathcal{N}(0,1)$ donc de moyenne 0 et variance 1. On va simuler N valeurs (1000 pour la première partie et on ira jusqu'à 1000000 de tirages) du sous-jacent en simulant un mouvement Brownien. D'où :

$$S_t(i) = S_0 e^{(r-\frac{\sigma^2}{2})T+\sigma\sqrt{T}Z_i}$$

Or, le prix du *CALL* européen avec la méthode de Monte Carlo est l'espérance qui grace à la loi des grands nombres est :

$$C_{MC} = \frac{1}{N}\sum_{i=1}^{N}(S_T(i) - K)_+$$

5.1 Analyse du CALL et du PUT

On présente dans cette section l'approximation du prix du CALL ou du PUT à partir de la simulation d'un mouvement Brownien avec la méthode de Monte-Carlo, la fonction permettant de le calculer est Listing 8 (L'implémentation se trouve dans le fichier MONTECARLO.PY). Les Figure 18, et Figure 19 on observe que la vitesse de convergence est plus lente. En effet l'amplitude des oscillations est élevée même après un nombre de tirages dépassant 1 million. De plus, on a implementé une fonction qui donne une mesure de la vitesse de convergence de notre méthode (V_CONVERGENCE()), et cette dernière indique une valeur de $k = \frac{U_n - \text{prix théorique}}{U_{n-1} - \text{prix théorique}} \neq 0$, avec n grand et des résultats moyénnés sur plusieurs lancements de la fonction.

De plus on on a tracé deux droites, représentant le rayon de convergence que devrait avoir notre méthode. Sachant que la méthode de Monte-Carlo est une aproximation en $O(\frac{1}{\sqrt{(n)}})$. Il est intéressant d'analyser le fait qu'en réalisant un développement limité des variations, on acumule une erreur trop importante. La fonction qui implemente cette a était laissé avec le label optionnel et non efficient dans le fichier MONTECARLO.PY.

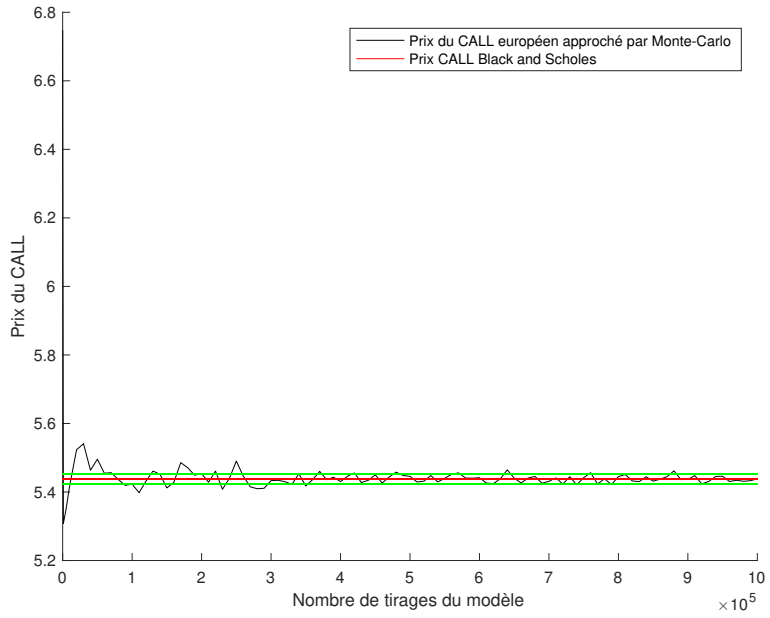

FIGURE 18 – Variation du prix du CALL européen approché par la méthode de Monte-Carlo en fonction du nombre de tirages

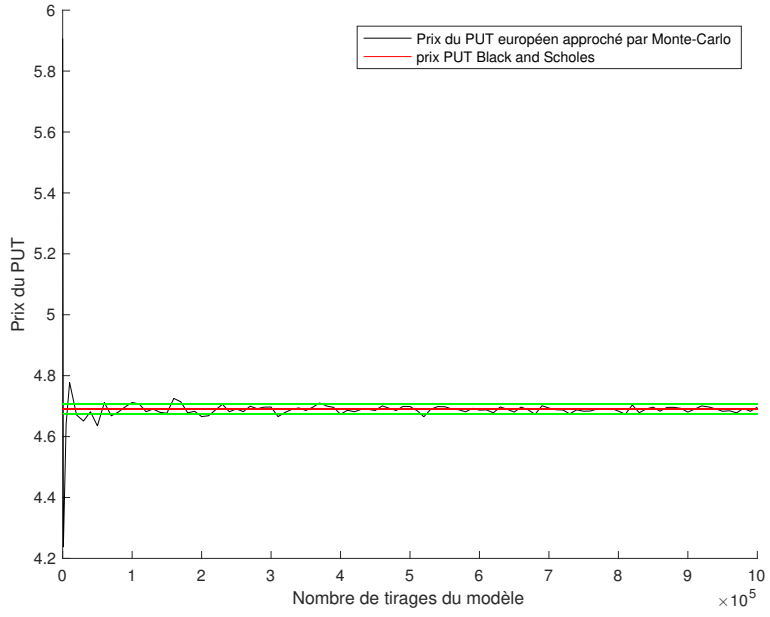

FIGURE 19 – Variation du prix du PUT européen approché par la méthode de Monte-Carlo en fonction du nombre de tirages

5.2 Analyse pour le delta

Pour ce qui est du δ on obtiens des résultats similaires en terme de convergence et de vitesse de convergence. Les variations et leur amplitude sont identiques, seulement les échelles peuvent varier en raison des fortes variances dans les faibles nombres de tirages. La fonction qui permet d'obtenir les résultats est Listing 9, et on peut la tester dans le fichier DELTA.PY.

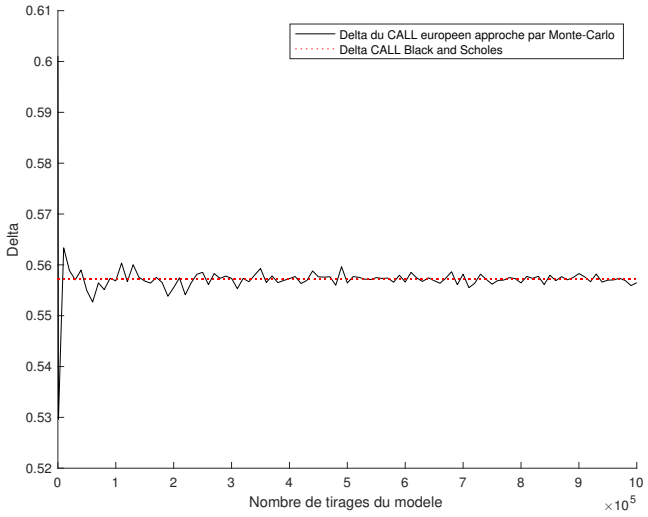

FIGURE 20 – Variation du δ d'un CALL européen approché par la méthode de Monte-Carlo en fonction du nombre de tirages

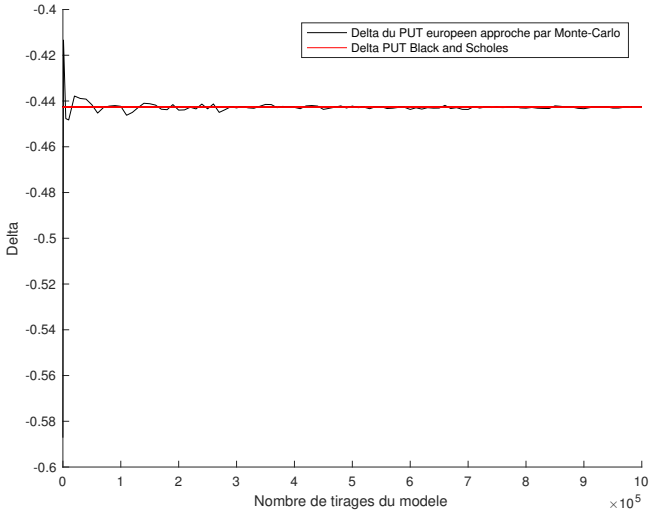

FIGURE 21 – Variation du δ d'un PUT européen approché par la méthode de Monte-Carlo en fonction du nombre de tirages

6 Option barrière

Cette option s'annule lorsque la barrière est franchie, elle est considerée comme exotique. Elle est une option plus sûre pour le vendeur car elle met une limite aux pertes potentiellement infinies sinon. Comme c'est une option plus risquée pour l'acheteur le prix de l'option est inférieur à celui d'une option vanille traditionelle. On la connait comme un CALL *up and out*. De plus on modelisera aussi un PUT avec barrière pour comprendre, comment implémenter la barrière et avoir une vision globale sur les 2 options. Le PUT avec barrière, aussi connu comme PUT *down and out* possède un barrière inférieure. On fera l'analyse avec une barrière à 65. Les Figure 22 et Figure 23 représentant le payoff de cette option en $t = T$ en fonction du cours du sous-jacent, S_t.

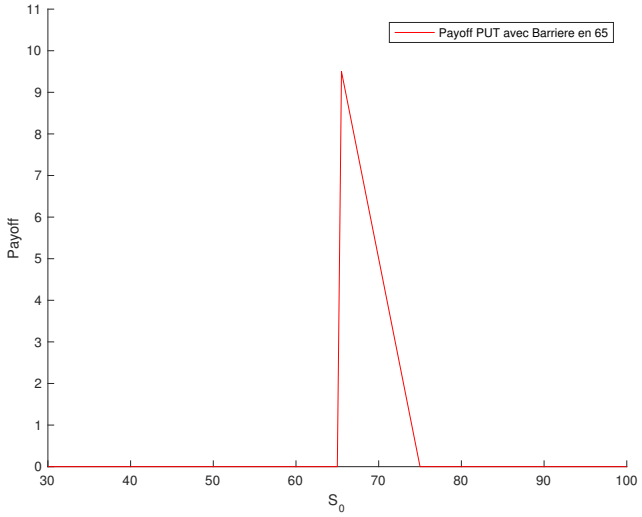

FIGURE 22 – Payoff d'une option barrière sur un PUT européen en fonction du cours du sous-jacent

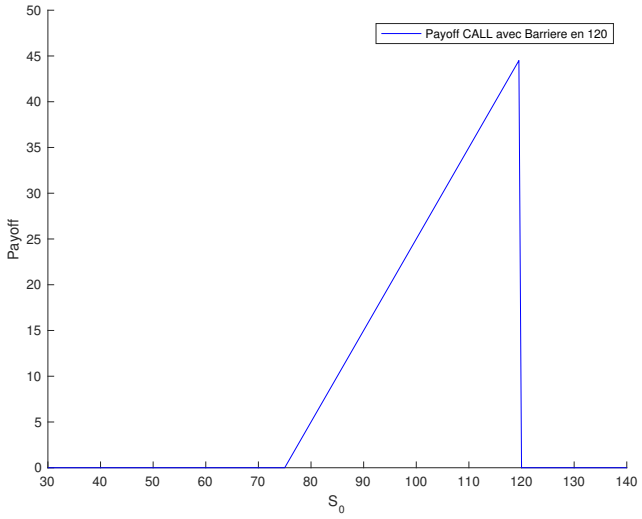

FIGURE 23 – Payoff d'une option barrière sur un CALL européen en fonction du cours du sous-jacent

6.1 Méthode de Monte-Carlo

On analyse le prix de l'option en réalisant l'approximation de Monte-Carlo. La fonction pour le réaliser est Listing 10, le fichier dans lequel on la implémentée est OPTION_BARRIERE.PY. Les Figure 24, Figure 25 présentent les résultats obtenus.

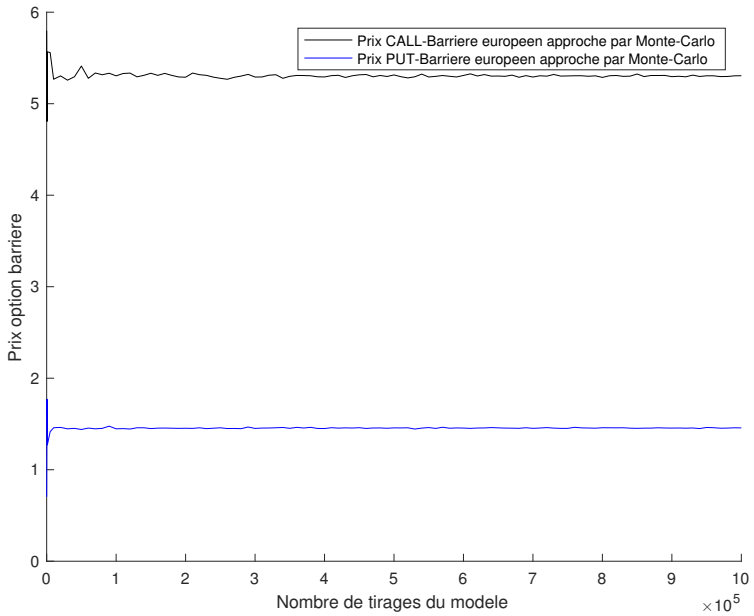

FIGURE 24 – Variation d'une option européenne avec option barrière approché par la méthode de Monte-Carlo en fonction du nombre de tirages

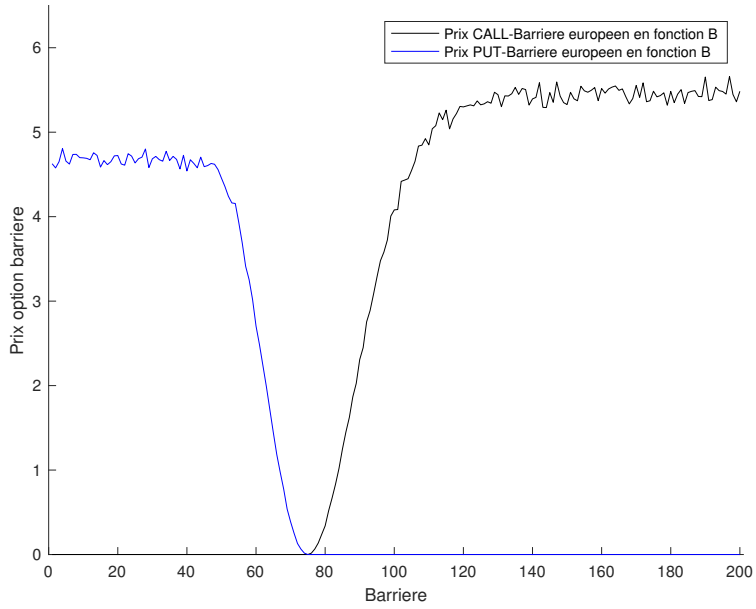

FIGURE 25 – Variation d'une option européenne avec option barrière approché par la méthode de Monte-Carlo en fonction du niveau de la barrière

On peut voir que le prix du CALL augmente avec le niveau de la barriere et pour un barriere très grande, le prix est celui du modèle Black and Scholes. Pour le Put c'est exactement le contraire. Ce qui est logique, en effet si on pose la barrière très en dessous du prix *At the Money* dans le cas d'un CALL, il est presque impossible que l'on ai un gain. Un raisonement similaire peut se faire pour le PUT.

On voit que le prix du PUT est inferieur au prix du CALL, car l'espérance de gain est inférieure.

6.2 Modèle Binomial

On analyse le prix de l'option dans le cadre du modèle binomial. La fonction pour le réaliser est Listing 11, le fichier dans lequel on la implémentée est OPTION_BARRIERE.PY. Les Figure 26, Figure 27 présentent les résultats obtenus.

Il est intéressant d'analyser comme sur la Figure 26 il y a plus de variations du PUT. La raison apparait sur la Figure 27, en effet on peut observer que pour un niveau de la barrière à 65, le delta du PUT est plus élevé que celui du CALL pour un niveau égal à 120.

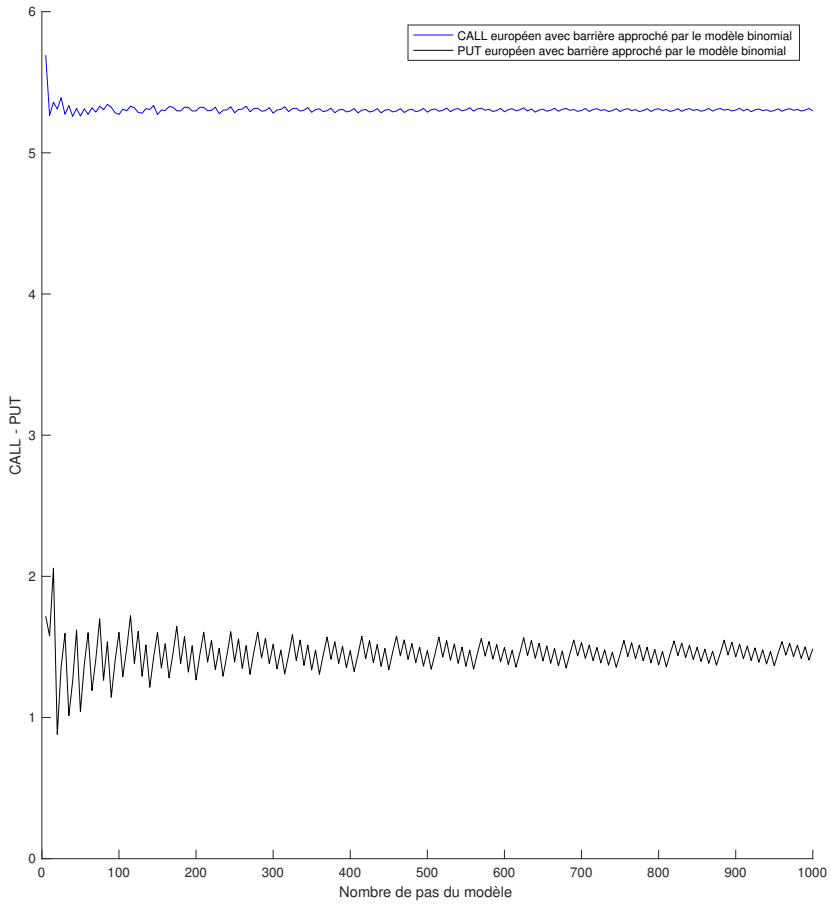

FIGURE 26 – Variation d'une option européenne avec option barrière approché par le modèle binomial en fonction du pas

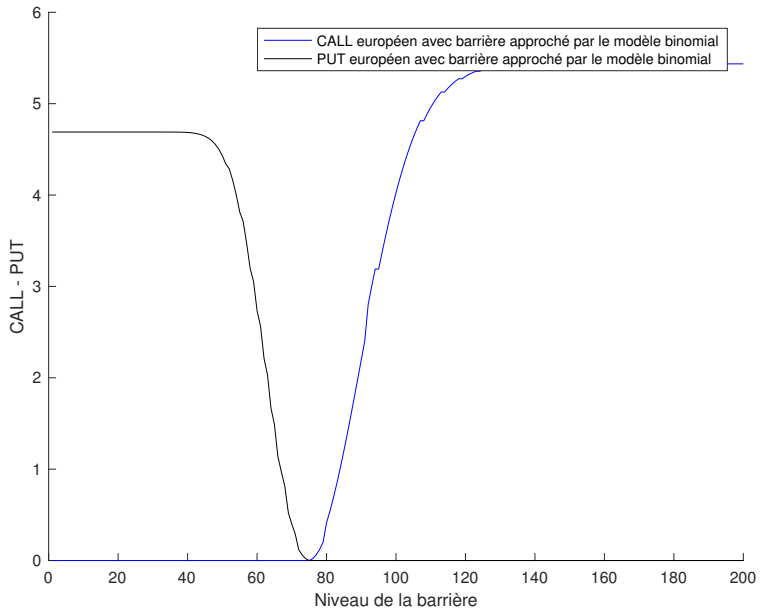

FIGURE 27 – Variation d'une option européenne avec option barrière approché par le modèle binomial en fonction du niveau de la barrière

Les remarques sur les variations en fonction du PUT son les mêmes que dans le cadre Monte-Carlo. De plus on observe un lissage plus important, en effet la méthode de Monte-Carlo nécessite plus de tirages pour obtenir un résultat équivalent au modèle binomial. Cependant le grand avantage de la méthode de Monte-Carlo est que l'on n'a pas besoin de connaitre le modèle pour obtenir des résultats.

7 Codes

```
%----------------------------
%   IMPLANTATION DU CALL
%----------------------------
S_0=75;K=75;T=1;sigma=0.17;r=0.01;K_0=K*exp(-r*T);

d_pos=(log(S_0/K)+(r+sigma^2/2)*T)/(sigma*T^0.5);
d_neg=(log(S_0/K)+(r-sigma^2/2)*T)/(sigma*T^0.5);

CALL = S_0*normcdf(d_pos)-K_0*normcdf(d_neg);
PUT = -S_0*normcdf(-d_pos)+K_0*normcdf(-d_neg);

%----------------------------
%   CALL EN FONCTION DE S0
%----------------------------
CALL_S0=[];
S_list=1:0.5:150;
for i = 1:size(S_list,2)
    CALL_S0(i) = S_list(i) * normcdf((log(S_list(i)/K) + (r+(sigma^2)/2)*T) /
    ↪  (sigma*T^0.5)) - K_0 * normcdf((log(S_list(i)/K) + (r-(sigma^2)/2)*T) /
    ↪  (sigma*T^0.5));
end
%----------------------------
%   CALL EN FONCTION DE SIGMA
%----------------------------
CALL_sigma=[];
sigma_list=0:1/1000:0.5;
for i = 1:size(sigma_list,2)
    CALL_sigma(i) = S_0 * normcdf((log(S_0/K) + (r+(sigma_list(i)^2)/2)*T) /
    ↪  (sigma_list(i) * (T^0.5))) - K_0 * normcdf((log(S_0/K) +
    ↪  (r-(sigma_list(i)^2)/2)*T) / (sigma_list(i)*(T^0.5)));
end
%----------------------------
%   CALL EN FONCTION DE T
%----------------------------
CALL_t=[];
T_list=0:0.001:4;
for i = 1:size(T_list,2)
    CALL_t(i) = S_0*normcdf((log(S_0/K) + (r+(sigma^2)/2)*T_list(i)) /
    ↪  (sigma*(T_list(i))^0.5)) - K*exp(-r*T_list(i)) * normcdf( (log(S_0/K) +
    ↪  (r-(sigma^2)/2)*T_list(i)) / (sigma*(T_list(i))^0.5));
end
```

Listing 1 – Calcul du CALL et du PUT via la méthode de Black and Scholes. Matlab.

```
%----------------------------
%        DONNÉES
%----------------------------
S_0 = 75; K = 75; T = 1; sigma = 0.17; R = 0.01; k2 = 80; k1 = 75;
%----------------------------
%   DELTA RISK REVERSAL
%----------------------------
DELTA_RR=[];
S_list=1:0.5:150;
for i = 1:size(S_list,2)
    DELTA_RR(i) = normcdf((log(S_list(i)/k1) +
    ↪  (R+(sigma^2/2))*T)/(sigma*sqrt(T))) - 1 - normcdf((log(S_list(i)/k2) +
    ↪  (R+(sigma^2/2))*T) / (sigma*sqrt(T)));
end
%----------------------------
%   GAMMA RISK REVERSAL
%----------------------------

GAMMA_RR=[];
S_list=1:0.5:150;
for i = 1:size(S_list,2)
    GAMMA_RR(i) = normpdf((log(S_list(i)/k1)+ (R+(sigma^2/2))*T)/
    ↪  (sigma*sqrt(T))) / (S_list(i) *sigma * sqrt(T) ) - normpdf((
    ↪  log(S_list(i)/k2)+ (R+(sigma^2/2))*T) /
    ↪  (sigma*sqrt(T)))/(S_list(i)*sigma*sqrt(T));
end
%----------------------------
%   VEGA RISK REVERSAL
%----------------------------
VEGA_RR=[];
S_list=1:0.5:150;
for i = 1:size(S_list,2)
    VEGA_RR(i) = normpdf((log(S_list(i)/k1) + (R+(sigma^2/2))*T) /
    ↪  (sigma*sqrt(T))) * S_list(i)* sqrt(T) - normpdf((log(S_list(i)/k2) + (R
    ↪  + (sigma^2/2)) * T) /(sigma*sqrt(T)))*S_list(i) * sqrt(T);
end
%----------------------------
%   THETA RISK REVERSAL
%----------------------------
THETA_RR=[];
S_list=1:0.5:150;
for i = 1:size(S_list,2)
    THETA_RR(i) = normpdf((log(S_list(i)/k1) +( R+ (sigma^2/2))*T) /
    ↪  (sigma*sqrt(T)))* S_list(i)*sigma / (2*sqrt(T)) + R*k1 * exp(-R*T) *
    ↪  (normcdf((log(S_list(i)/k1) + (R-(sigma^2/2))*T)/(sigma*sqrt(T)))-1) -
    ↪  normpdf((log(S_list(i)/k2) + (R+(sigma^2/2))*T) / (sigma*sqrt(T))) *
    ↪  S_list(i)*sigma / (2*sqrt(T))+ R*k2*exp(-R*T) *
    ↪  normcdf((log(S_list(i)/k2) + (R-(sigma^2/2))*T))/(sigma*sqrt(T));
end
```

Listing 2 – Calcul des grecques du CALL dans le modèle de Black et Scholes

```
"""Fonction de simulation de calcul des valeurs a utiliser"""

def valeurs(sigma,n,t,r,T) :
    delta=T/float(n-t)
    u=math.exp(math.sqrt(delta)*sigma)
    d=1/u
    R=math.exp(r*T/(n-t))
    p=(R-d)/(u-d)
    return u,d,p,delta
```

Listing 3 – Fonction de calcul des valeurs nécessaires au modèle binomial

```
"""Fonction de simulation du resultat d'un arbre binomial """

def arbre(S,n,p,u):
    S_arbre=[]
    u,d,p,delta=valeurs(sigma,n,t,r,T) #obtention des valeurs
    for i in range(1,n+1):
        if rd.binomial(1, p, 1)==True :
            S_arbre.append(S*u)
            S=S*u
        else:
            S_arbre.append(S*d)
            S=S*d
    return S_arbre,S #on obtiens un simulation du "chemin" prit par S
```

Listing 4 – Simulation du payoff aleatoire dans le modèle binomial

```
"""Calcul du prix d'une option par avec le modele binomial"""

#type prend les valeurs CALL ou PUT
#style prend les valeurs EURO ou AMER

def modele_binomial(S0, K, r, sigma, T, N,type,style):

    # initialisation calcul des valeurs
    u,d,p,delta=valeurs(sigma,n,t,r,T) # obtention des valeurs du modele
    # initialisation de l'arbre
    prix = [[0.0 for j in xrange(i + 1)] for i in xrange(N + 1)]
    # Calcul prix finaux en fonction du type d'option
    if type=="CALL":
        for j in xrange(i+1):
            prix[N][j] = max(S0 * u**j * d**(N - j)-K, 0.0)
        if style=="AMER":
            for i in xrange(N-1, -1, -1):
                for j in xrange(i + 1):
                    prix[i][j] = math.exp(-r * delta) * (p * prix[i + 1][j + 1]
                    ↪  + (1.0 - p) * prix[i + 1][j])
                    exercise = S0 * u**j * d**(i - j)-K
                    prix[i][j] = max(prix[i][j],exercise)
        else :
            for i in xrange(N-1, -1, -1):
                for j in xrange(i + 1):
                    prix[i][j] = math.exp(-r * delta) * (p * prix[i + 1][j + 1]
                    ↪  + (1.0 - p) * prix[i + 1][j])
    else :
        for j in xrange(i+1):
            prix[N][j] = max(K- S0 * u**j * d**(N - j), 0.0)
        if style=="AMER":
            for i in xrange(N-1, -1, -1):
                for j in xrange(i + 1):
                    prix[i][j] = math.exp(-r * delta) * (p * prix[i + 1][j + 1]
                    ↪  + (1.0 - p) * prix[i + 1][j])
                    exercise = K-S0 * u**j * d**(i - j)
                    prix[i][j] = max(prix[i][j],exercise)
        else :
            for i in xrange(N-1, -1, -1):
                for j in xrange(i + 1):
                    prix[i][j] = math.exp(-r * delta) * (p * prix[i + 1][j + 1]
                    ↪  + (1.0 - p) * prix[i + 1][j])
    return prix[0][0]
```

Listing 5 – Algorithme de calcul du prix d'une option via le modèle binomial

```
"""Calcul de la frontiere d'exercice d'une option americaine """

def frontiere(S0, K, r, sigma, T, N,type):
    # initialisation calcul des valeurs
    u,d,p,delta=valeurs(sigma,n,t,r,T)
    # initialisation de l'arbre
    valeur_exercice=[] ; temps=[] ; stock=[]
    prix = [[0.0 for j in xrange(i + 1)] for i in xrange(N + 1)]
    # Calcul prix finaux en fonction du type d'option
    if type=="CALL":
        for j in xrange(i+1):
            prix[N][j] = max(S0 * u**j * d**(N - j)-K, 0.0)
        for i in xrange(N-1, -1, -1):
            exercice_iter=[] ; temps_iter=[] ; stock_iter=[]
            m=False
            for j in xrange(i + 1):
                prix[i][j] = math.exp(-r * delta) * (p * prix[i + 1][j + 1]+
                ↪  (1.0 - p) * prix[i + 1][j])
                exercise = S0 * u**j * d**(i - j)-K
                if max(prix[i][j],exercise)==exercise :
                    m=True
                    stock_iter.append(exercise)
                    exercice_iter.append(j)
                    temps_iter.append(i)
            if m==True:
                valeur_exercice.append(max(exercice_iter))
                temps.append(max(temps_iter))
                stock.append(max(stock_iter))
    else :
        for j in xrange(i+1):
            prix[N][j] = max(K- S0 * u**j * d**(N - j), 0.0)
        for i in xrange(N-1, -1, -1):
            exercice_iter=[] ; temps_iter=[] ; stock_iter=[]
            m=False
            for j in xrange(i + 1):
                prix[i][j] = math.exp(-r * delta) * (p * prix[i + 1][j + 1]+
                ↪  (1.0 - p) * prix[i + 1][j])
                exercise = K-S0 * u**j * d**(i - j)
                if max(prix[i][j],exercise)==exercise :
                    m=True
                    stock_iter.append(exercise)
                    exercice_iter.append(j)
                    temps_iter.append(i)
            if m==True:
                valeur_exercice.append(max(exercice_iter))
                temps.append(max(temps_iter))
                stock.append(S0 * u**max(exercice_iter) * d**(max(temps_iter) -
                ↪  max(exercice_iter)))
    return stock,temps
```

Listing 6 – Calcul de la frontiere d'exercice d'une option americaine (temps,S_t)

```
"""Calcul du delta d'une option par recurrence/modele binomial"""

def delta_df(delta_S,type):
    delta = (modele_binomial(S+ delta_S, K, r, sigma, T,n,type,"EURO") -
    ↪  modele_binomial(S- delta_S, K, r, sigma, T,n,"EURO",style)) /
    ↪  (2*delta_S)
    return delta
```

Listing 7 – Algorithme de calcul du delta d'une option via le modèle binomial

```
"""Calcul du prix d'une option europeenne de part la simulation de Monte-Carlo"""
def Option_euro_MC(S0, K, r, sigma, T, type,tirages):
    # initialisation du vecteur prix
    prix=0
    #generation des variations aleatoires suivant une loi normale autour de S,
    ↪  avec la fonction scipy.stats.norm.rvs qui nous donne l'option d'obtenir
    ↪  les valeurs aleatoires de la loi normale
    for i in xrange(tirages):
        S_t = S0*math.exp(T*(r-0.5*sigma**2))
         #generation des epsilon aleatoires suivant une loi normale
        epsilon = scipy.stats.norm.rvs(size=1)
        #variations autour de S en simulant un mouvement Brownien
        S_t = S_t*math.exp(epsilon*(T*sigma**2)**0.5)
        if type=="CALL":
            prix += max(S_t - K, 0.0)
        else :
            prix += max(K-S_t, 0.0)
    prix=prix/(tirages)*math.exp(-r*T)
    return prix #obtention de la moyenne de mouvements aleatoires (Browniens)
```

Listing 8 – Algorithme de calcul du prix d'une option européenne via la simulation de Monte-Carlo

```
""" Calcul du delta d'une option europeene de part la
methode de Monte-Carlo"""

def delta_MC(S0, K, r, sigma, T, type, n, tirages,delta_S):
    # initialisation du vecteur prix
    prix_up = 0
    prix_down = 0

    #generation des variations aleatoires suivant une loi normale autour de S,
    ↪ avec la fonction scipy.stats.norm.rvs qui nous donne l'option d'obtenir
    ↪ les valeurs aleatoires de la loi normale
    for i in range(tirages):
        S_up = S0+delta_S
        S_down = S0-delta_S
        #generation des epsilon aleatoires suivant une loi normale
        epsilons = scipy.stats.norm.rvs(size=1)
        #On calcule l'evolution du sous-jacent avec un differentiel de
        ↪ difference
        S_up = S_up * math.exp((r-(sigma**2)/2)*T + sigma*math.sqrt(T) *
        ↪ epsilons)
        S_down = S_down * math.exp((r-(sigma**2)/2) * T + sigma*math.sqrt(T) *
        ↪ epsilons)
        if type=="CALL":
            prix_up += max(S_up - K, 0.0)
            prix_down += max(S_down - K, 0.0)
        else :
            prix_up += max(K-S_up, 0.0)
            prix_down += max(K-S_down, 0.0)

    p_up=scipy.mean(prix_up)*math.exp(-r*T)
    p_down=scipy.mean(prix_down)*math.exp(-r*T)
    greek_delta=float(p_up-p_down)/float(2*delta_S)
    return greek_delta/tirages
```

Listing 9 – Algorithme de calcul du delta d'une option européenne via la simulation de Monte-Carlo

```python
"""Calcul du prix d'une option euro avec option barriere par simulation de Monte-Carlo"""

def Option_euro_BAR_MC(S0, K, r, sigma, T, type, tirages,OPT_BAR):
    # initialisation de prix et S_t
    prix = 0
    S_t = S0
    #Calcul constant x pour ameliorer efficience de l'algorithme
    x=(r-0.5*sigma**2)*T
    if type=="CALL":
        for i in range(tirages):
            epsilons = scipy.stats.norm.rvs(size=1)
            S_t = S0*math.exp(x+sigma*math.sqrt(T)*epsilons)
            if S_t < OPT_BAR :   #condition barriere CALL
                prix += max(S_t - K, 0)
            else :
                prix += 0
    else :
         for i in range(tirages):
            epsilons = scipy.stats.norm.rvs(size=1)
            S_t = S0*math.exp(x+sigma*math.sqrt(T)*epsilons)
            if S_t > OPT_BAR :  #condition barriere PUT
                prix += max(K-S_t, 0)
            else :
                prix += 0

    return math.exp(-r*T)*prix/tirages #obtention de la moyenne
```

Listing 10 – Algorithme de calcul du prix d'une option barrière via la simulation de Monte-Carlo

```
"""Calcul du prix d'une option barriere (CALL/PUT) par recurrence"""

def option_BAR_rec(S0, K, r, sigma, T, N,OPT_BAR):

    # intialisation calcul des valeurs
    u,d,p,delta=valeurs(sigma,n,t,r,T)
    # initialisation de l'arbre
    prix = [[0.0 for j in xrange(i + 1)] for i in xrange(N + 1)]
    if type=="CALL":
    # Calcul prix finaux
        for j in xrange(i+1):
            if S0 * u**j * d**(N - j)<OPT_BAR:
                prix[N][j] = max(S0 * u**j * d**(N - j)-K, 0.0)
            else :
                prix[N][j]=0
    else:
        for j in xrange(i+1):
            if S0 * u**j * d**(N - j)>OPT_BAR:
                prix[N][j] = max(K-S0 * u**j * d**(N - j), 0.0)
            else :
                prix[N][j]=0
    # Recurrence
    for i in xrange(N-1, -1, -1):
        for j in xrange(i + 1):
            prix[i][j] = math.exp(-r * delta) * (p * prix[i + 1][j + 1] + (1.0
            ↪  - p) * prix[i + 1][j])
    return prix[0][0]
```

Listing 11 – Algorithme de calcul du prix d'une option barrière via le modèle binomial