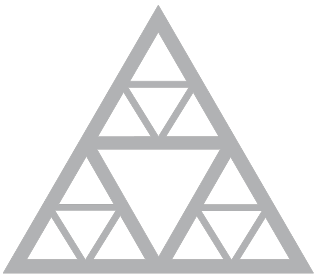

École des Ponts
ParisTech

Calcul d'un put américain et frontière d'exercice

Projet de méthodes mathématiques
pour la finance

Mohammed Amine Kheldouni
Victor Marchais

30 mai 2017

Table des matières

Remerciements

Nous tenons tout d'abord à remercier Monsieur Bernard Lapeyre, pour le temps qu'il nous a accordé et pour la pertinence de ses conseils. Les échanges que nous avons pu réaliser avec lui ont été d'une importance capitale dans l'avancement et la réussite de notre projet.

1 Introduction

Dans ce projet, on cherche une approximation de la frontière d'exercice du put américain par une exponentielle. Cette approximation, faite par Nengjiu Ju, constitue en effet une méthode d'évaluation du put américain.

On se place dans ce qui suit, dans le cadre du modèle de Black-Scholes sans dividendes. Nous allons dans un premier temps, établir la théorie mathématique du modèle d'approximation de la frontière d'exercice du put américain. Ensuite, nous verrons les résultats d'une simulation numérique de cette approximation pour retrouver les hyperparamètres du modèle de Ju.

2 Théorie mathématique

2.1 Etudes de l'EDP de Black-Scholes

Dans la zone d'exercice, nous avons une formule explicite du prix du put américain étudier :

$$\forall x \leq s(t), \quad u(t,x) = (K-x)^+ = K - x$$

Par suite, en recalculant les termes de l'EDP de Black-Scholes dans la zone d'exercice on obtient le résultat suivant :

$$\begin{array}{rcl} \frac{\partial u}{\partial t} + \frac{\sigma^2 x^2}{2}\frac{\partial^2 u}{\partial x^2} + rx\frac{\partial u}{\partial x} - ru & = & -rx - ru(t,x) \\ & = & -rK \end{array}$$

Donc dans la zone d'exercice (en dessous de la frontière d'exercice), le prix du put américain est régi par l'EDP suivante :

$$\frac{\partial u}{\partial t} + \frac{\sigma^2 x^2}{2}\frac{\partial^2 u}{\partial x^2} + rx\frac{\partial u}{\partial x} - ru = -rK$$

2.2 Construction d'une martingale

Soit $M_t = e^{-rt}u(t,S_t) + K\int_0^t re^{-rv}\mathbb{1}_{(S_v \leq s(v))}dv$

Rappelons que sous la probabilité risque neutre $\mathbb{P}^*$ le processus W_t est un mouvement brownien. De plus, dans le cadre du modèle de Black Scholes S_t se réécrit en fonction de W_t comme suit :

$$\widetilde{S}_t = \widetilde{S}_0 e^{\sigma W_t - \frac{\sigma^2}{2}t}$$

et $d\widetilde{S}_t = \sigma\widetilde{S}_t dW_t$. Il vient alors que :

$$dS_t = d(e^{rt}\widetilde{S}_t = e^{rt}(r\widetilde{S}_t dt + d\widetilde{S}_t)$$

Soit en réintégrant le terme d'actualisation dans l'actif risqué actualisé, on obtient

$$dS_t = \sigma S_t dW_t + rS_t dt$$

Par ailleurs, la fonction $u : (t,x) \to u(t,x)$ est dans $C^{1,2}([0,T[\times\mathbb{R}_+^*)$. D'après la formule d'Itô appliquée à cette fonction :

$$d(u(t,S_t)) = \frac{\partial u}{\partial t}(t,S_t)dt + \frac{\partial u}{\partial x}(t,S_t)dS_t + \frac{1}{2}\frac{\partial^2 u}{\partial x^2}(t,S_t)d < S >_t$$

avec $d < S >_t = \sigma^2 S_t^2 dt$.

On utilise donc la formule d'Itô écrite ci-dessus lors de la différentiation de $e^{-rt}u(t,S_t)$. On obtient alors l'équation suivante :

$$\begin{aligned} d(e^{-rt}u(t,S_t)) &= -re^{-rt}u(t,S_t)dt + e^{-rt}d(u(t,S_t)) \\ &= e^{-rt}(\tfrac{\partial u}{\partial t}(t,S_t)dt + \tfrac{\partial u}{\partial x}(t,S_t)(\sigma S_t dW_t + rS_t dt) + \tfrac{\sigma^2 S_t^2}{2}\tfrac{\partial^2 u}{\partial x^2}(t,S_t)dt - rudt) \\ &= e^{-rt}(-rK)\mathbb{1}_{(S_t \leq s(t))} - e^{-rt}S_t\tfrac{\partial u}{\partial x}(t,S_t)\sigma dW_t \end{aligned}$$

Puis en isolant le second terme du membre de droite, on fait apparaître la différentielle de M_t, soit :

$$dM_t = -e^{-rt}S_t\frac{\partial u}{\partial x}(t,S_t)\sigma dW_t$$

Par cette formule, et sous réserve que $\mathbb{E}(\int_0^T (e^{-rt}S_t\frac{\partial u}{\partial x}(t,S_t)\sigma)^2 dt < \infty$, on aura montré que M_t est bien une $\mathbb{P}^*$ martingale.
Il faut donc maintenant montrer que

$$\mathbb{E}(\int_0^T (e^{-rt}S_t\frac{\partial u}{\partial x}(t,S_t)\sigma)^2 dt < \infty$$

Or, comme $\frac{\partial u}{\partial x}$ est bornée

$$\exists C > 0/ \quad (e^{-rt}S_t\frac{\partial u}{\partial x}(t,S_t)\sigma)^2 \leq CS_t^2$$

De plus, S_t est un processus régi par l'équation différentielle stochastique de Black-Scholes, donc S_t est bien de carré intégrable.

Par suite

$$\mathbb{E}(\int_0^T (e^{-rt}S_t\frac{\partial u}{\partial x}(t,S_t)\sigma)^2 dt < \infty$$

Et donc M_t est une martingale.

2.3 Formule de prime d'exercice anticipé

De la sous-section précédente, nous pouvons écrire la formule de la martingale en fin d'échéance (à $t = T$). Il vient donc que :

$$M_T = e^{-rT}u(T,S_T) + K\int_0^T re^{-rv}\mathbb{1}_{(S_v \leq s(v))}dv$$

De plus, un passage en espérance dans cette formule fait apparaître $u(0,S_0)$. En effet, comme $(M_t)_t$ est une martingale :

$$\mathbb{E}(M_T) = \mathbb{E}(M_0) = u(0,S_0)$$

Par ailleurs, le terme $e^{-rT}u(T,S_T)$ exprime bien le payoff actualisé et donc la valeur du put américain au temps initial.
Soit

$$e^{-rT}u(T,S_T) = P(0,S_0)$$

Enfin, en considérant la fonction $v \to re^{-rv}\mathbb{1}_{(S_v \leq s(v))}$, on remarque que celle-ci est positive. On peut donc appliquer le théorème de Fubini sur le dernier terme de la formule et on obtient l'égalité suivante :

$$\mathbb{E}(\int_0^T re^{-rv}\mathbb{1}_{(S_v \leq s(v))}dv) = \int_0^T re^{-rv}\mathbb{E}(\mathbb{1}_{(S_v \leq s(v))})dv$$

En regroupant nos calculs précédents, on trouve bien la formule de prime d'exercice anticipée :

$$u(0,S_0) = P(0,S_0) + K\int_0^T re^{-rv}\mathbb{E}(\mathbb{1}_{(S_v \leq s(v))})dv$$

2.4 Approximation de la frontière d'exercice

On considère donc l'approximation de Ju de la frontière d'exercice s en une exponentielle. Le problème devient alors :
Trouver A et a pour que s soit approchée "le mieux" par la fonction $t \to Ae^{at}$.

Pour une telle fonction la formule de prime d'exercice anticipée devient :

$$\begin{aligned} u_J(0,S_0) &= P(0,S_0) + K\int_0^T re^{-rt}\mathbb{E}^*(\mathbb{1}_{(S_t \leq Ae^{at})})dt \\ &= P(0,S_0) + K\int_0^T re^{-rt}\mathbb{P}^*(S_t \leq Ae^{at})dt \end{aligned}$$

Or $S_t = S_0e^{\sigma W_t - \frac{\sigma^2}{2}t}$. Donc :

$$\begin{aligned} u_J(0,S_0) &= P(0,S_0) + K\int_0^T re^{-rt}\mathbb{P}^*(S_0e^{\sigma W_t - \frac{\sigma^2}{2}t} \leq Ae^{at})dt \\ &= P(0,S_0) + K\int_0^T re^{-rt}\mathbb{P}^*(W_t \leq \frac{1}{\sigma}(ln(\frac{A}{S_0}) + (\frac{\sigma^2}{2}+a)t))dt \end{aligned}$$

Par ailleurs, W_t est un $\mathbb{P}^*$ mouvement brownien donc suit une loi normale de variance t. Par suite $W_t = \sqrt{t}\mathcal{N}(0,1)$ (égalité en loi)
Par suite, en divisant par $\sqrt{t}$ dans la probabilité, on fait apparaître la fonction de répartition de la loi normale centrée réduite N. D'où

$$u_J(0,S_0) = P(0,S_0) + K\int_0^T re^{-rt}N((\frac{\sigma}{2}+\frac{a}{\sigma})\sqrt{t} + \frac{\frac{ln(\frac{A}{S_0})}{\sigma}}{\sqrt{t}})dt$$

Par identification :

$$b_1 = \frac{\sigma}{2} + \frac{a}{\sigma} \qquad b_2 = \frac{ln(\frac{A}{S_0})}{\sigma}$$

2.5 Formalisation du problème

Soit $b_3 = \sqrt{b_1^2 + 2r}$ et considérons la fonction f suivante :

$$\begin{aligned} f : t \quad \rightarrow \quad & -e^{-rt}N(b_1\sqrt{t} + \tfrac{b_2}{\sqrt{t}}) + \tfrac{1}{2}(\tfrac{b_1}{b_3} + 1)e^{b_2(b_3-b_1)}N(b_3\sqrt{t} + \tfrac{b_2}{\sqrt{t}}) + \\ & \tfrac{1}{2}\left(\tfrac{b_1}{b_3} - 1\right)e^{-b_2(b_3+b_1)}N(b_3\sqrt{t} - \tfrac{b_2}{\sqrt{t}}) \end{aligned}$$

Cette fonction est dérivable en t pour un temps stirctement positif par composée et somme de fonctions dérivables. De plus, le calcul de f' se fait comme suit :

$$\begin{aligned} f'(t) \quad = \quad & re^{-rt}N(b_1\sqrt{t} + \tfrac{b_2}{\sqrt{t}}) - e^{-rt}(\tfrac{b_1}{2\sqrt{t}} - \tfrac{b_2}{2t\sqrt{t}})\tfrac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}(b_1\sqrt{t}+\frac{b_2}{\sqrt{t}})^2} \\ & +\tfrac{1}{2}(\tfrac{b_1}{b_3} + 1)e^{b_2(b_3-b_1)}(\tfrac{b_3}{2\sqrt{t}} - \tfrac{b_2}{2t\sqrt{t}})\tfrac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}(b_3\sqrt{t}+\frac{b_2}{\sqrt{t}})^2} \\ & +\tfrac{1}{2}(\tfrac{b_1}{b_3} - 1)e^{-b_2(b_3+b_1)}(\tfrac{b_3}{2\sqrt{t}} + \tfrac{b_2}{2t\sqrt{t}})\tfrac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}(b_3\sqrt{t}-\frac{b_2}{\sqrt{t}})^2} \end{aligned}$$

Ensuite, en développant le contenu des exponentielles, les termes se simplifient pour ne laisser que le tout premier terme de dérivation. On trouve alors après coût la formule de f' :

$$\forall t \in]0, T], \quad f'(t) = re^{-rt}N(b_1\sqrt{t} + \frac{b_2}{\sqrt{t}})$$

On en déduit une nouvelle expression pour la prime d'exercice approchée :

$$u_J(0, S_0) = P(0, S_0) + K(f(T) - f(0)) \qquad (*)$$

en notant $f(0) = \lim_{t\to 0} f(t)$
$(*)$ est une expression analytique de $u_J(0, S_0)$.

Dans la prochaine section, nous proposons une méthode de résolution numérique basée sur cette expression analytique pour déterminer A et a correspondant le mieux à la frontière d'exercice.

3 Simulations numériques

3.1 Modélisation du problème

On se place dans le cadre du modèle de Black-Scholes avec des valeurs numériques comme suit :

Tableau numérique des valeurs considérées

σ	0.05
r	0.08
S_0	\$20
K	\$22
T	6 mois

Nous avons implémenté notre modèle d'approximation de la frontière d'exercice pour évaluer le put américain par la méthode de Nengjiu JU avec le langage PYTHON. Notre but étant de

Trouver (A, a) tels que

$$\begin{cases} u_J(0,A) &= (K-A)^+ \\ \frac{\partial u_J}{\partial x}(0,A) &= -1 \end{cases}$$

Nous avons donc créé un moteur (classe PYTHON) *"ApproxFrontier"* qui par ses méthodes effectue le calcul des fonctions importantes dont on a besoin pour résoudre le système.

3.2 Optimisation et résolution de système

Pour résoudre ce système numérique a priori non linéaire, nous avons pensé à utiliser une librairie de résolution de système comme SYMPY. Cette librairie se base sur des méthodes matricielles de décomposition LU pour retrouver les paramètres recherchés. Malheureusement, cette méthode n'a pas abouti. La résolution aboutissait à la formation de matrices singulières.

Nous avons dans un second temps, reformuler le système à résoudre pour le convertir en algorithme d'optimisation (type point fixe, Newton, ...). On a en effet réussi à effectuer le calcul de la dérivée $\frac{\partial f}{\partial A}(t)$ car la dépendance de la fonction f en A n'apparaît que dans la formule de b_2. Mais le calcul des autres dérivées premières du Jacobien étant assez long, nous avons préféré effectuer une résolution par dichotomie.

3.3 Résultats

Avec les données numériques énoncées précédemment, nous avons trouver les valeurs suivantes pour la frontière d'exercice :

$$\begin{cases} A &= \$\,16.6 \\ a &= 0.21 \end{cases}$$

En réinjectant ces valeurs dans le système pour valider celles-ci, on trouve :

$$u_J(0, A) = (K - A)_+ = \$\,5.397$$

Et

$$\frac{\partial u_J}{\partial x}(0, A) = -1$$

L'erreur sur la valeur trouvée est finalement de l'ordre de $\epsilon = 10^{-3}$.

4 Conclusion

En somme, nous avons réussi à redémontrer la formule de prime d'exercice anticipée, sous certaines hypothèses, à savoir que $u \in C^{1,2}([0, T[\times\mathbb{R}_+^*)$ et que $\frac{\partial u}{\partial x}$ est bornée.
Toutefois, la formule de prime d'exercice anticipé peut être justifiée rigoureusement, sans ces hypothèses simplificatrices.

Nous avons finalement réussi à proposer une approximation exponentielle de la frontière d'exercice, en trouvant A et a, et en évaluant le put américain.

Appendice

```
from sympy import *

from numpy import *
import matplotlib.pyplot as plt


class ApproxFrontier:
    def __init__(self):
        #Constantes
        self.sigma = 0.05
        self.r = 0.08
        self.mu = 0.1
        self.s0 = 20
        self.K = 1.1*self.s0
        self.T =0.5

    def N(self, x):
        """fonction de r partition de la loi normale centr e r duite
           (= probabilit  qu'une variable al atoire distribu e selon
           cette loi soit inf rieure    x)
           formule simplifi e propos e par Abramovitz & Stegun dans le livre
           "Handbook of Mathematical Functions" (erreur < 7.5e—8)
        """
        u = abs(x) # car la formule n'est valable que pour x>=0

        Z = 1/(sqrt(2*pi))*exp(—u*u/2) # ordonn e de la LNCR pour l'absisse u

        b1 = 0.319381530
        b2 = —0.356563782
        b3 = 1.781477937
        b4 = —1.821255978
        b5 = 1.330274429

        t = 1/(1+0.2316419*u)
        t2 = t*t
        t4 = t2*t2
```

```
    P = 1—Z*(b1*t + b2*t2 + b3*t2*t + b4*t4 + b5*t4*t)
    if x < 0:
        P = 1.0—P # traitement des valeurs x<0

    return P

def gaussian(self, x):
    return 1/sqrt(2*pi) * exp(—x**2/2)

def b1(self, a):
    return self.sigma/2 + a/self.sigma

def b2(self, A):
    return log(A/self.s0)/self.sigma

def b3(self, a):
    return sqrt(self.b1(a)**2 + 2*self.r)

def f(self, A,a,t):
    b_1 = self.b1(a)
    b_2 = self.b2(A)
    b_3 = self.b3(a)
    t1 = —exp(—self.r*t)*self.N(b_1*sqrt(t)+ b_2/(sqrt(t)))
    t2 = (1/2)*(b_1/b_3 +1)*exp( b_2*(b_3—b_1))*self.N(b_3*sqrt(t) \
    + b_2/sqrt(t) )
    t3 = (1/2)*(b_1/b_3 —1)*exp(—b_2*(b_1+b_3))*self.N(b_3*sqrt(t) — \
    b_2/sqrt(t) )
    return t1+t2+t3

def df_dx(self, A,a,t):
    b_1 = self.b1(a)
    b_2 = self.b2(A)
    b_3 = self.b3(a)
    db2 = 1/(self.sigma*A)
    t1 = —exp(—self.r*t)/sqrt(t) * self.gaussian(b_1*sqrt(t)+b_2/sqrt(t))
    t2 = 1/2 * (b_1/b_3 + 1) * exp(b_2*(b_3—b_1)) * ((b_3—b_1) * \
    self.N(b_3*sqrt(t)+b_2/sqrt(t))+1/sqrt(t) * \
    self.gaussian(b_3*sqrt(t)+b_2/sqrt(t)))
    t3 = 1/2 * (b_1/b_3 — 1) * exp(—b_2*(b_3+b_1)) * (—(b_3+b_1) * \
```

```
    self.N(b_3*sqrt(t)-b_2/sqrt(t))-1/sqrt(t) * \
    self.gaussian(b_3*sqrt(t)-b_2/sqrt(t)))
    return (t1+t2+t3)*db2


def d1(self, t):
    return ( log(self.s0/self.K)+ t*(self.r+self.sigma**2/2) )/\
    self.sigma*sqrt(t)

def d2(self, t):
    return self.d1(t) - self.sigma*sqrt(t)

def P(self, X,t):
    dd2 = self.d2(t)
    dd1 = self.d1(t)
    return -X * self.N(-dd1) + self.K * exp(-self.r*t) * self.N(-dd2)

def dP0(self,X):
    dd2 = self.d2(0.00001)
    dd1 = self.d1(0.00001)
    return -self.N(-dd1)+X/(X*self.sigma*sqrt(self.T))*self.N(-dd1) + \
    self.K*exp(-self.r*0)*self.N(-dd2)

def u_dx(self,A,a):
    return self.dput(A) + self.K * self.df_dx(A,a,self.T) - \
    self.K*self.df_dx(A,a,0.00001)

def u(self,A,a):
    res = self.put(A) + self.K*(self.f(A,a,self.T) -self.f(A,a,0.00001))
    return res


def put(self, A):
    if A > self.K:
        return 0
    else:
        return (self.K-A)

def dput(self, A):
```

```
        if A > self.K:
            return 0
        else:
            return -1

    def u_optimize(self, A, a):
        return self.u(A,a) - self.put(A)

    def du_optimize(self, A, a):
        return self.u_dx(A,a) - self.dput(A)

    def solver(self):
        A = Symbol('A')
        a = Symbol('a')
        nsolve((u_optimize,du_optimize), (A,a), (20, 0.5))


M = ApproxFrontier()

A = 16.6033
print("Prix du put americain :")
print(M.K-A)
a = 0.210308108108
print("Test valeur de l'approximation de Ju : ")
print(M.u(A,a))
print("Valeur de la d riv e en (A,a) :")
print(M.u_dx(A,a))
print("################")

a = 0.210308108108

# A = linspace(9,200,5000)
# #a = linspace(0.7,10,1000)
# Yu = [M.u_optimize(x,a) for x in A]
# Yud = [M.du_optimize(x,a) for x in A]
# plt.plot(A,Yu)
# plt.plot(A,Yud)
# plt.show()
```

```
"""
A = Symbol('A', real=True)
a = Symbol('a', real=True)
result = nsolve((M.u_optimize(A,a), M.du_optimize(A,a)), (A, a), (1, 1))
"""
```
