Finance Quantitative

Modélisation du smile

Patrick Hénaff

Version: 30 mars 2023

In this problem set, you will use the following functions:

GBSPrice: Price of a vanilla option:

$$P = f(\text{PutCall}, S, K, T, r, b, \sigma)$$

where:

PutCall 'c' for a call, 'p' for a put

b cost of carry: ridk free rate r less dividend yield d

r risk-free rate

```
GBSPrice <- function(PutCall, S, K, T, r, b, sigma) {
  d1 <- (log(S/K) + (b+sigma^2/2)*T)/(sigma*sqrt(T))
  d2 <- d1 - sigma*sqrt(T)

  if(PutCall == 'c')
    px <- S*exp((b-r)*T)*pnorm(d1) - K*exp(-r*T)*pnorm(d2)
  else
    px <- K*exp(-r*T)*pnorm(-d2) - S*exp((b-r)*T)*pnorm(-d1)

px
}
```

GBSVega: Vega ($\frac{\partial P}{\partial \sigma}$) of a Vanilla option:

```
GBSVega <- function(PutCall, S, K, T, r, b, sigma) {
  d1 <- (log(S/K) + (b+sigma^2/2)*T)/(sigma*sqrt(T))
  S*exp((b-r)*T) * dnorm(d1)
}
```

Data

The spot is $S = 110$. We observe the following prices for calls and puts on an asset paying a continuous dividend.

Strike	Call	Put
70	38.496	0.017
75	33.656	0.055
80	28.863	0.143
85	24.193	0.336
90	19.722	0.736
95	15.493	1.395
100	11.704	2.499
105	8.529	4.213
110	5.851	6.390
115	3.831	9.279
120	2.372	12.702
125	1.374	16.542
130	0.768	20.823
135	0.414	25.315
140	0.208	29.994
145	0.093	34.765
150	0.049	39.594
155	0.020	44.445
160	0.008	49.309

```
df <- read.csv("../GP/data/call-put-prices.csv")
S0<- 110
T <- 0.5

kable(df[, c("Strike", "Call", "Put")], booktabs=TRUE, digits=c(0,3,3)) %>%
  kable_styling()
```

Questions

Implied dividend yield and risk-free rate

Using the Call-Put parity, estimate by linear regression the implied risk-free rate (r) and dividend yield (d).

Solution

On utilise la relation de parité call-put au temps t, pour des options de maturité T sur une option payant un dividende continu d:

$$C_t - P_t = S_t e^{-d(T-t)} - Ke^{-r(T-t)}$$

avec:

C_t prix du call en t

P_t prix du put en t

S_t Spot en t

d taux de dividende continu

r taux sans risque

T Date d'expiration

Du fait des erreurs de mesure, la relation n'est pas vérifiée exactement, mais on peut estimer les termes $e^{-d(T-t)}$ and $e^{-r(T-t)}$ par regression:

$$C_t - P_t = a_0 + a_1 K$$

Ce qui donne les expressions suivantes pour r et d:

$$\begin{aligned} r = & \quad -\frac{1}{T}\ln(-a_1) \\ d = & \quad \frac{1}{T}\ln\left(\frac{S_t}{a_0}\right) \end{aligned}$$

```
T <- 0.5
mod <- lm(Call- Put ~ Strike, data=df)
a0 <- as.numeric(mod$coefficients[1])
a1 <- as.numeric(mod$coefficients[2])
r.hat <- -log(-a1)/T
d.hat <- log(S0/a0)/T
```

On obtient les estimations suivantes: $r = 5\%$ and $d = 6.01\%$.

Implied Volatility calculation

1. Using the functions above, write a function that computes the implied volatility of a Vanilla option. Let:
$$g(\sigma) := P - f(\text{PutCall}, S, K, T, r, b, \sigma)$$

where P is the observed price of the option of interest.

We look for the volatility σ such that $g(\sigma) = 0$.

The function should have the following signature:

```
ImpliedVol <- function(p, TypeFlag, S, X, Time, r, b, sigma=NULL, maxiter=500, tol=1.e-5) {
}
```

where:

p price of the option

sigma an optional initial value for the volatility

maxiter an optional maximum number of iterations

tol an optional tolerance for the error $|g(\sigma)|$.

Solution

La méthode de Newton génère une sequence monotone convergeant vers la solution si l'algorithme utilise pour point initial la valeur:

$$\sigma_0 = \sqrt{\frac{2|\ln(F/K)|}{T}}$$

Ce qui donne l'algorithme suivant:

1. Initialiser $\sigma_0 = \sqrt{\frac{2|\ln(F/K)|}{T}}$
2. Tant que $|C(\sigma_n) - C^*| > \epsilon$:
    1. Calculer
    $$\sigma_{n+1} = \sigma_n + \frac{C^* - C(\sigma_n)}{\frac{\partial C}{\partial \sigma}}$$
    2. $n \leftarrow n + 1$

Mise en oeuvre:

```
ImpliedVol.J <- function(p, TypeFlag, S, X, Time, r, b, tol, maxiter=50) {
  # prix à terme
  F <- S * exp((r-b)*T)
  s <- sqrt(2*abs(log(F/X))/T)
  not_converged <- T
  vega <- GBSVega(TypeFlag, S, X, Time, r, b, s)
  i <- 1
  while(not_converged & (i<maxiter)) {
    err <- (p-GBSPrice(TypeFlag, S, X, Time, r, b, s))
    s <- s + err/vega
    not_converged <- (abs(err/vega) > tol)
    if(s<0) not_converged <- TRUE
    i <- i+1
  }

if(not_converged) NaN else s

}
```

On met en évidence l'intérêt de choisir le point de changement de convéxité comme point initial en comparant le temps de calcul selon la méthode proposée et selon la méthode standard de la librairie fOptions:

```
TypeFlag <- 'c'
S <- 100
X <- 100
Time <- 1
r <- .03
b <- .01
sigma <- .314
tol <- 1e-6
n = 1000
p <- GBSPrice(TypeFlag, S, X, Time, r, b, sigma)
```

On répète n fois le calcul:

```
  t1 <- function(n) {
    start <- Sys.time()
  si <- sapply(seq(n), function(i){GBSVolatility(p, TypeFlag, S, X, Time, r, b, tol=tol, maxiter=50)})
  end <- Sys.time()
  clock_time <- as.numeric(end-start, units="secs")
  c(mean(si), sd(si), clock_time)}

  t2 <- function(n) {
    start <- Sys.time()
  si <- sapply(seq(n), function(i){ImpliedVol.J(p, TypeFlag, S, X, Time, r, b, tol)})
  end <- Sys.time()
  clock_time <- as.numeric(end-start, units="secs")
  c(mean(si), sd(si), clock_time)}
```

Les résultats sont résumés dans le tableau ci-dessous.

	σ	$sd(\sigma)$	Durée(sec)
GSBVolatility	0.314	0	2.065
Mod. Newton	0.314	0	0.058

2. Test the accuracy of your procedure on options that are deep in the money and deep out of the money, and report the results of your tests.

Solution

```
maxiter=50
test_iv <- function(K) {
  p <- GBSPrice(TypeFlag, S, K, Time, r, b, sigma)
  iv.1 <- GBSVolatility(p, TypeFlag, S, K, Time, r, b, tol, maxiter)
  iv.2 <- ImpliedVol.J(p, TypeFlag, S, K, Time, r, b, tol, maxiter)
  c(iv.1, iv.2)
}

sigma <- .31
K.io <- c(seq(40, 80, by=10), seq(140, 200, by=10))
sigma.io <- sapply(K.io, test_iv)
```

The résultat montre que la méthode de Newton n'est pas robuste pour les strikes très loins du strike ATM. Une méthode du premier ordre qui n'utilise pas le vega, telle que l'algorithme de bissection, donne des résultats satisfaisants:

```
ImpliedVolBisection <- function(p, TypeFlag, S, X, Time, r, b,
                                tol, sBounds, maxiter=100) {
sMin <- min(sBounds)
sMax <- max(sBounds)
pMin <- GBSOption(TypeFlag, S, X, Time, r, b, sMin)@price
pMax <- GBSOption(TypeFlag, S, X, Time, r, b, sMax)@price
```

```
not_converged <- abs(pMin-pMax) > tol
i <- 1

while(not_converged & (i<maxiter)) {
  sStar <- (sMin + sMax)/2
  pStar <- GBSOption(TypeFlag, S, X, Time, r, b, sStar)@price
  if(pStar < p) {
    pMin <- pStar;
    sMin <- sStar
  } else {
    pMax <- pStar;
    sMax <- sStar
  }

  not_converged <- (abs(pMin-pMax) > tol)
  i <- i+1
}

ifelse(not_converged, NaN, (sMin+sMax)/2)
}
```

```
test_iv_b <- function(K) {
  p <- GBSPrice(TypeFlag, S, K, Time, r, b, sigma)
  iv.2 <- ImpliedVolBisection(p, TypeFlag, S, K, Time, r, b, tol, sigma.bounds, maxiter)
  iv.2
}

sigma <- .31
sigma.bounds <- c(.1, .6)
K.io <- c(seq(40, 80, by=10), seq(140, 200, by=10))
sigma.io.2 <- sapply(K.io, test_iv_b)
```

Table 1: Vol Implicite ITM et OTM

	σ_{BS}	σ_{Newton}	σ_{Bis}
40	0.31001	NaN	0.31002
50	0.31000	NaN	0.31000
60	0.31000	0.31	0.31000
70	0.31000	0.31	0.31000
80	0.31000	0.31	0.31000
140	0.31000	0.31	0.31000
150	0.31000	0.31	0.31000
160	0.31000	0.31	0.31000
170	0.31000	0.31	0.31000
180	0.31000	0.31	0.31000
190	0.31000	0.31	0.31000
200	0.31000	NaN	0.31000

3. Compute the implied volatility of the calls and puts in the data set.

Solution

On reprend les données du problème:

```
r <- NULL
b <- NULL
tol <- 1.e-7
maxiter <- 100
b.hat <- r.hat - d.hat
sigma.bounds <- c(.1, .5)
iv.calc <- function(K, C.price, P.price) {
  iv.c <- ImpliedVolBisection(C.price, TypeFlag='c', S0, K, T, r.hat, b.hat, tol, sigma.bounds, maxiter)
  iv.p <- ImpliedVolBisection(P.price, TypeFlag='p', S0, K, T, r.hat, b.hat, tol, sigma.bounds, maxiter)
  #iv.c <- ImpliedVol.J(C.price, TypeFlag='c', S0, K, T, r.hat, b.hat, tol, maxiter)
  #iv.p <- ImpliedVol.J(P.price, TypeFlag='p', S0, K, T, r.hat, b.hat, tol, maxiter)
  c(iv.c, iv.p)
}

res <- mapply(iv.calc, df$Strike, df$Call, df$Put)
```

Les volatilités implicites des calls et puts sont:

Table 2: Vol Implicite des Calls et Puts

K	σ_C	σ_P
70	0.24496	0.23724
75	0.23595	0.23350
80	0.22762	0.22759
85	0.22412	0.22225
90	0.22057	0.21900
95	0.21344	0.21300
100	0.20879	0.20915
105	0.20758	0.20833
110	0.20280	0.20278
115	0.19985	0.20095
120	0.19697	0.19844
125	0.19363	0.19356
130	0.19194	0.19256
135	0.19100	0.18881
140	0.18923	0.18709
145	0.18580	0.18473
150	0.18790	0.18425
155	0.18499	0.18096
160	0.18298	0.17083

On remarque des disparités dans les volatilités des calls et puts de strikes faibles. Or, du fait de la parité call-put, les volatilités des calls et puts de même strike doivent être identiques. On attribue ces disparités à un manque de liquidité pour les options de strike éloigné du prix du sous-jacent.

4. Fit a quadratic function to the call and put implied volatilities (one function for the calls, one for the puts), and plot actual vs. fitted data. Interpret the results.

Solution

```
iv.c <- res[1,]
iv.p <- res[2,]
lm.c <- lm(iv.c ~ poly(df$Strike,2,raw=TRUE))
lm.p <- lm(iv.p ~ poly(df$Strike,2,raw=TRUE))
smileVol <- function(coef, K) {
  sum(coef * c(1, K, K*K))
}
```

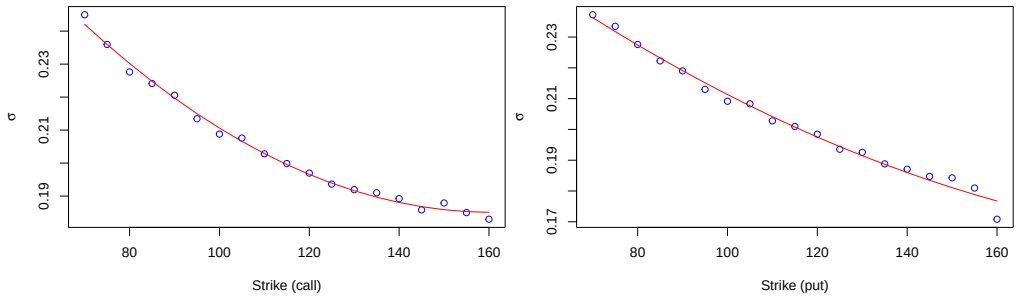


Figure 1: Volatilité implicite

On observe des erreurs plus importantes pour les options très "dans l'argent". On peut attribuer ce phénomène au fait que le marché est plus liquide pour les options "à l'argent"ou un peu "hors de l'argent".

Breeden-Litzenberger formula

Compute the implied density of S_T using the Breeden-Litzenberger formula. Estimate

$$\frac{\partial^2 f}{\partial K^2}$$

by finite difference. Remember that now σ is a function of strike. Plot the implied distribution and compare to the distribution implicit in the standard Black-Scholes model. Interpret your observations.

Solution

On estime le modèle quadratique sur l'ensemble des données:

```
lm.strike <- lm(c(iv.c, iv.p) ~ poly(rep(df$Strike,2), 2, raw = TRUE))
  b1 <- lm.strike$coefficients[1]
  b2 <- lm.strike$coefficients[2]
  b3 <- lm.strike$coefficients[3]
  smileVol <- function(K) {
       b1 + b2 * K + b3 * K^2
   }

bsVol <- function(K) median(c(iv.c, iv.p))
```

La densité de S_T selon le modèle log-normal est:

```
bs.pdf <- function(S, K, T, b, sigma) {
   d1 <- (log(S/K) + (b + sigma^2/2) * T)/(sigma * sqrt(T))
   d2 <- d1 - sigma * sqrt(T)
   sT <- sqrt(T)
   dnorm(d2)/(K * sigma * sT)
}
```

Calcul de $\frac{\partial^2 f}{\partial K^2}$ par différence finie et de la densité de S_T compte-tenu du smile:

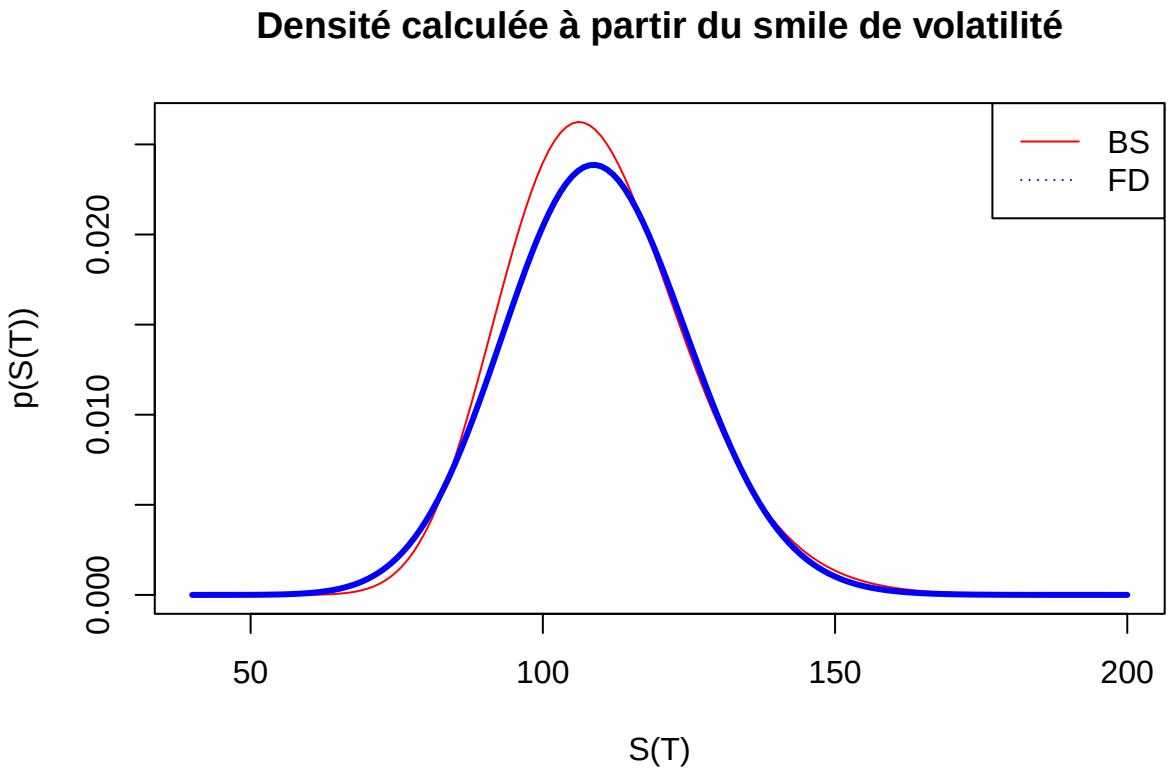
```
d2CdK2 <- function(vol, S, K, T, r, b) {
   dK <- K/10000
   c <- GBSPrice('c', S, K, T, r, b, vol(K))
   cPlus <- GBSPrice('c', S, K+dK, T, r, b, vol(K+dK))
   cMinus <- GBSPrice('c', S, K-dK, T, r, b, vol(K-dK))
   (cPlus-2*c+cMinus)/(dK^2)
}


smile.pdf <- function(S, K, T, r, b) {
  d2CdK2(smileVol, S, K, T, r, b) * exp(r*T)
}
```

On normalise la fonction densité pour que l'intégrale de la densité soit égale à 1. L'erreur selon la formule de Breeden-Litzenberger est non-négligeable.

```
sigma.BS <- median(c(iv.c, iv.p))
p.bs <- function(K) {
  bs.pdf(S0, K, T, b.hat, sigma.BS)
}

p.smile <- function(K){
  smile.pdf(S0, K, T, r.hat, b.hat)
}

sum.bs <- integrate(Vectorize(p.bs),
  lower=10, upper=700)$value
sum.smile <- integrate(Vectorize(p.smile),
  lower=10, upper=700)$value

print(c(sum.bs, sum.smile))
```

```
## [1] 1.000000 1.062082
```

Les densités en T selon les deux modèles montrent que la probabilité d'une forte baisse implicite dans le prix des options est supérieur à celle induite par le modèle log-normal. On note également l'assymétrie de la distribution empirique.



Pricing a digital call

Recall that a digital call with strike K pays one euro if $S_T \geq K$, and nothing otherwise.

Using the implied density computed above, compute the price of a digital call by numerical integration.

Compare with the price obtained using a log-normal distribution for S_T. Interpret your observations.

Solution

Valeur à maturité d'un call digital:

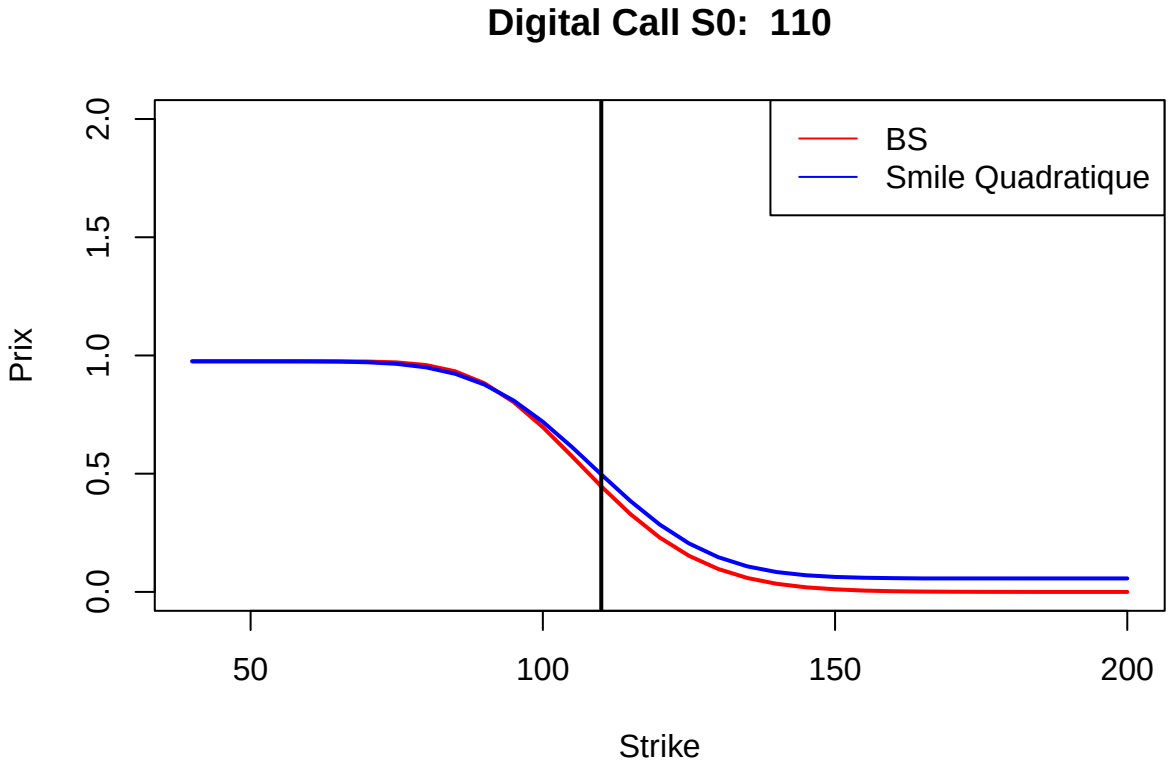
```
digital.call.payoff <- function(S) {
  if(S>Kd)
    1
  else
    0
}

digital.put.payoff <- function(S) {
  if(S > Kd)
    0
  else
    1
}

sigma.BS <- median(c(iv.c, iv.p))
KdRange <- seq(40, 200, 5)
```

```
BS.Price <- vector('numeric', length=length(KdRange))
Smile.Price <- vector('numeric', length=length(KdRange))
```

Fonctions à intégrer numériquement:

```
digital.bs <- function(K) {
  digital.call.payoff(K)*bs.pdf(S0, K, T, b.hat, sigma.BS)/sum.bs
}

digital.smile <- function(K){
  digital.call.payoff(K)*smile.pdf(S0, K, T, r.hat, b.hat)/sum.smile
}

for(i in seq_along(KdRange)) {
  Kd <- KdRange[i]

  BS.Price[i] <- exp(-r.hat*T)*integrate(
  Vectorize(digital.bs),
  lower=10, upper=700)$value

Smile.Price[i] <- exp(-r.hat*T)*integrate(
  Vectorize(digital.smile),
  lower=10, upper=700)$value
}
```

La figure ci-dessous illustre l'impact du smile sur le prix des calls et puts. L'effet de l'assymétrie de la distribution empirique est clairement visible.



```r
digital.bs <- function(K) {
  digital.put.payoff(K)*bs.pdf(S0, K, T, b.hat, sigma.BS)/sum.bs
}

digital.smile <- function(K){
  digital.put.payoff(K)*smile.pdf(S0, K, T, r.hat, b.hat)/sum.smile
}

for(i in seq_along(KdRange)) {
  Kd <- KdRange[i]

  BS.Price[i] <- exp(-r.hat*T)*integrate(
  Vectorize(digital.bs),
  lower=10, upper=700)$value

Smile.Price[i] <- exp(-r.hat*T)*integrate(
  Vectorize(digital.smile),
  lower=10, upper=700)$value
}
```