COS Method for European Option Valuation

The COS method is a numerical technique used in pricing of options in the case of absence of an explicit solution in addition to having the characteristic function. It is known of being fast even for multiple strikes which we need for model calibration.

Let X_t be a process such that $X_t = \ln(S_t/K)$, with S_t being the price of the underlying asset and K the strike price. In addition we assume that the process X takes values x and y at $t = t_0$ and $t = T$ respectively. Therefore, the value of a vanilla option is given by:

$$V(t_0, x) = e^{-r(T-t_0)} E[V(T, y)|\mathcal{F}_{t_0}] = e^{-r(T-t_0)} \int_R V(T, y) f_X(T, y, t_0, x) dy,$$

where:

- the expectation is being calculated under the risk-neutral measure,
- $f_X(T, y, t_0, x)$ is the transition probability density,
- r is the interest rate.

Next we truncate the infinite integration range to $[a, b]$ in R, the simple way of choosing that range (to be considered with caution) is to put $[a, b] = [-L\sqrt{T}, L\sqrt{T}]$, where L is a parameter that can change depending on the model being used, generally it takes 8 or 10. Hence,

$$V(t_0, x) \approx e^{-r(T-t_0)} \int_a^b V(T, y) f_X(T, y, t_0, x) dy.$$

Finally, using the density approximation by its Fourier cosine expansion we can write

$$V(t_0, x) \approx e^{-r(T-t_0)} \left\{ \frac{1}{2} F_0(x).H_0 + \sum_{k=1}^{N-1} F_k(x).H_k \right\}$$

- $H_k = \frac{2}{b-a} \int_a^b V(T, y) cos(k\pi \frac{y-a}{b-a}) dy$
- $F_k(x) = \frac{2}{b-a} Re\{\phi_X(\frac{k\pi}{b-a}, x; t_0, T).exp(-i\frac{ka\pi}{b-a})\}$
- ϕ_X is the characteristic function.

European options

Put $y = ln(\frac{S_T}{K})$. For European options, we have $V(T, y) = K(e^y - 1)^+$ for a call (resp. $K(1 - e^y)^+$ for a put) and

- H_k-coefficients for call: $H_k = \frac{2}{b-a} K(\psi_k(0, b) - \varphi_k(0, b))$
- H_k-coefficients for put: $H_k = \frac{2}{b-a} K(\varphi_k(a, 0) - \psi_k(a, 0))$

where

$$\psi_k(c, d) = \frac{1}{1 + \left(\frac{k\pi}{b-a}\right)^2} \left[cos\left(k\pi \frac{d-a}{b-a}\right) e^d - cos\left(k\pi \frac{c-a}{b-a}\right) e^c + \frac{k\pi}{b-a} sin\left(k\pi \frac{d-a}{b-a}\right) e^d - \frac{k\pi}{b-a} sin\left(k\pi \frac{c-a}{b-a}\right) e^c \right]$$

and

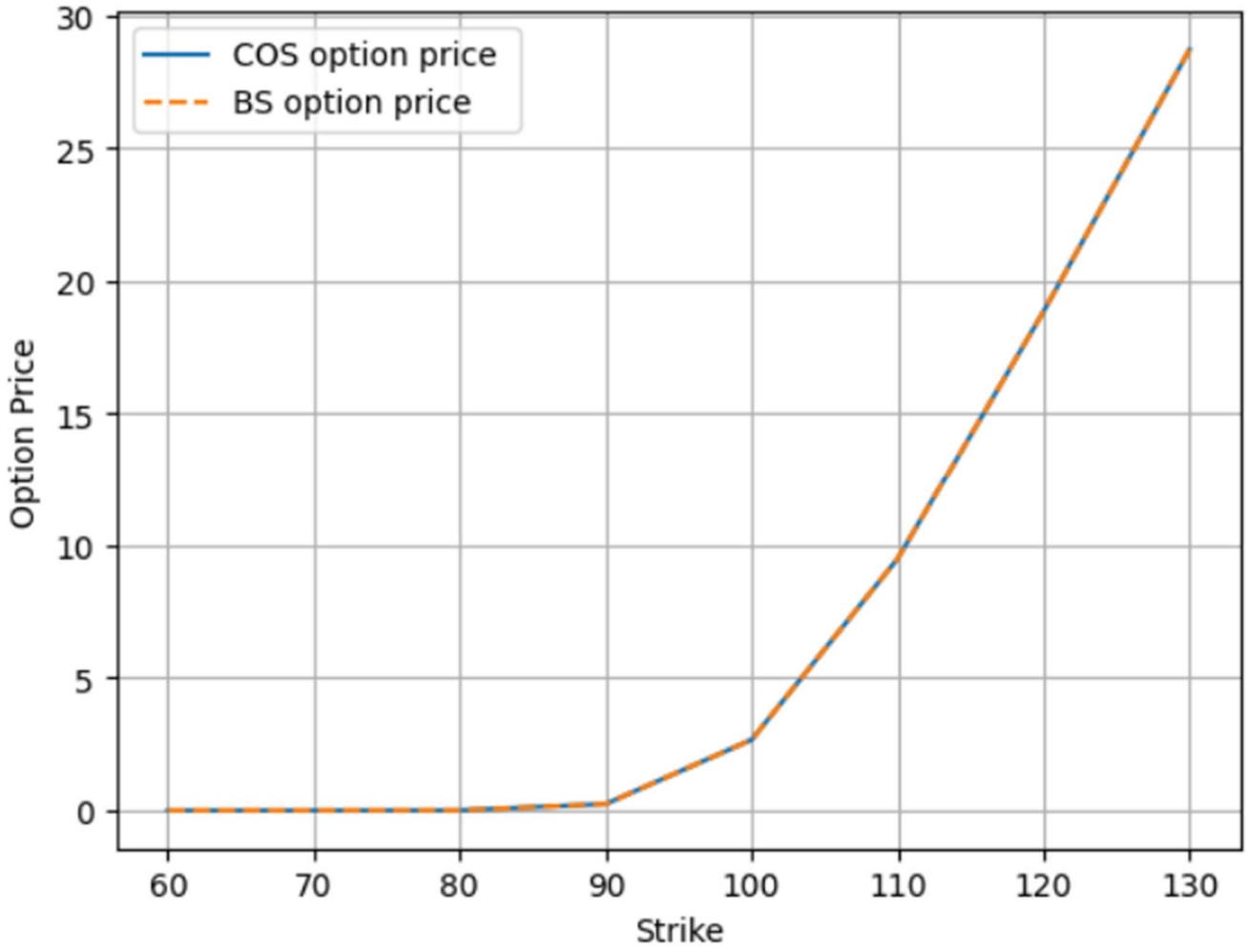
$$\varphi_k(c, d) = \frac{b-a}{k\pi} \left[sin\left(k\pi \frac{d-a}{b-a}\right) - sin\left(k\pi \frac{c-a}{b-a}\right) \right] 1_{k \neq 0} + (d - c) 1_{k=0}.$$

For more details about the demonstrations, the reader can check [1] pp. 169-173.

Code example for the GBM process

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as st

# COS method option price
def COSM(cf,CP,S0,r,tau,K,N,L):
    """
    The COS method for pricing European options its takes as arguments
    cf : characteristic function as a function
    CP   - C for call and P for put
    S0 : the initial price
    r : the rsik free rate
    tau : time to maturity (T-t_0)
    K : list of strikes
    N : Number of expansion terms
    L : the parameter for the truncation interval
    """
    K = np.array(K).reshape([len(K),1])  # reshape K to a column vector
    i = complex(0.0,1.0)
    x0 = np.log(S0 / K)
    k = np.linspace(0,N-1,N).reshape([N,1])
    a = -L * np.sqrt(tau)
    b = L * np.sqrt(tau)
    u = k * np.pi / (b - a)

    # Definition of the H_k coefficients
    Psi = lambda c, d: 1.0 / (1.0 + np.power((k * np.pi / (b - a)) , 2.0)) *
                 (np.cos(k * np.pi * (d - a)/(b - a)) * np.exp(d)
               - np.cos(k * np.pi * (c - a) / (b - a)) * np.exp(c)
               + k * np.pi / (b - a) * np.sin(k * np.pi * (d - a) / (b - a))
               - k * np.pi / (b - a) * np.sin(k * np.pi * (c - a) / (b - a)) *
                 np.exp(c))
    Phi = lambda c,d: np.sin(k * np.pi * (d - a) / (b - a))
                      - np.sin(k * np.pi *  (c - a)/(b - a))
    H_k = np.linspace(0,0,len(k)).reshape([len(k),1])

    # H_k for call and put
    if str(CP).upper()=="C":
            H_k[1:] = 2.0 / (b - a) * (Psi(0,b)[1:] - Phi(0,b)[1:] * (b - a) /
                      (k[1:] * np.pi))
            H_k[0] = 2.0 / (b - a) * (Psi(0,b)[0] - b)
    elif str(CP).upper()=="P":
            H_k[1:] = 2.0 / (b - a) * (Phi(a,0)[1:]* (b - a) / (k[1:] * np.pi)
                       - Psi(a,0)[1:])
            H_k[0] = 2.0 / (b - a) * (-a - Psi(a,0)[0])


    mat = np.exp(i * np.outer(- a , u) + i*np.outer(x0,u)) # "iux" is included here
```

```
    temp = cf(u) * H_k
    temp[0] = 0.5 * temp[0]
    value = np.exp(-r * tau) * K * np.real(mat.dot(temp))
    return value



#  Black Scholes option price

def BS_Option_Price(CP,S_0,K,sigma,tau,r):
    """
    Black-Scholes option price
    """
    K = np.array(K).reshape([len(K),1])
    d1    = (np.log(S_0 / K) + (r + 0.5 * np.power(sigma,2.0))
    * tau) / float(sigma * np.sqrt(tau))
    d2    = d1 - sigma * np.sqrt(tau)
    if str(CP).upper() == "C":
        value = st.norm.cdf(d1) * S_0 - st.norm.cdf(d2) * K * np.exp(-r * tau)
    elif str(CP).upper() == "P":
        value = st.norm.cdf(-d2) * K * np.exp(-r * tau) - st.norm.cdf(-d1)*S_0
    return value


# "iux" is included in the calculations above so we do not include it in the
definition of the characteristic function
# Hence the characteristic function in this case is of a normal distribution
# with parameters ((r - 0.5 *sigma**2)tau, tau*sigma**2)


i = complex(0.0,1.0)
CP = "P"
S0 = 100.0
r = 0.1
tau = 0.1
sigma = 0.25
K = [60.0, 70.0, 80.0, 90.0, 100.0, 110, 120.0, 130.0]
N = 128
L = 10
cf = lambda u: np.exp((r - 0.5 * np.power(sigma,2.0)) * i * u * tau - 0.5 *
np.power(sigma, 2.0) * np.power(u, 2.0) * tau)
COS_option_price = COSM(cf,CP,S0,r,tau,K,N,L)

# Compare the COS method price with the Black Scholes price

Exact_option_price = BS_Option_Price(CP,S0,K,sigma,tau,r)
plt.plot(K,COS_option_price)
plt.plot(K,Exact_option_price,'--')
plt.xlabel("Strike")
plt.ylabel("Option Price")
plt.legend(["COS option price ","BS option price "])
plt.grid()
```



[1] Oosterlee, Cornelis W., and Lech A. Grzelak. *Mathematical modeling and computation in finance: with exercises and Python and MATLAB computer codes*. World Scientific, 2019.