Derivatives Pricing Homework 4

Pan Yiming, Tai Lo Yeung, Qiwei Liu

2021/03/08

1 Pricing Asian options

Consider a one-year Asian call option struck at K with daily setting dates assuming the underlying stock price satisfies Black Scholes with parameters S_0, σ, r .

1.1 arithmetic and geometric average.

For the arithmetic average, we compute it by the formula:

$$A(T) = \frac{1}{N} \sum_{i=1}^{N} S_{ti}$$

For the geometric average, we compute it by the formula:

$$G(T) = (\prod_{i=1}^{N} S_{ti})^{1/N}$$

We simulate one path of stock price and then compute the arithmetic and geometric average. The plot is as following. We can see from the picture that the value of geometric average is very close to arithmetic average. But geometric average is a little lower than arithmetic average.

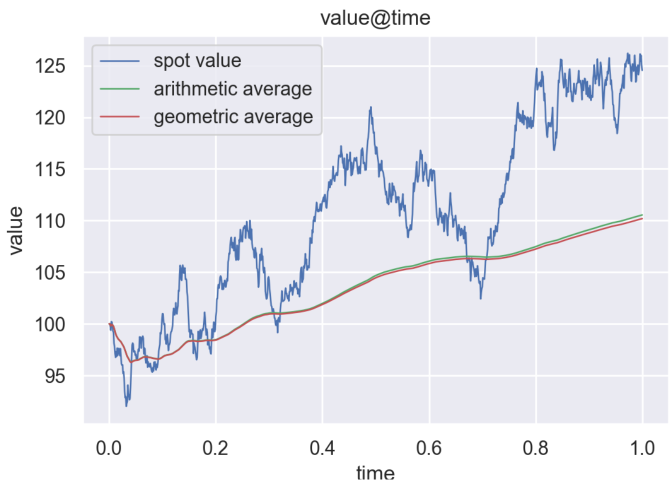

Figure 1: arithmetic and geometric average

1.2 Asian call value at time 0

We set $r = 0.05, K = 105, S_0 = 100, \sigma = 0.20, N = 1000, T = 1.$

For the Asian call value computed by arithmetic average, we use the following formula:

$$C^A_{tj} = e^{-r(T-t)}\frac{1}{n}\sum_{k=1}^{n} max(\frac{1}{N+1}(\sum_{i=0}^{j} S_{ti} + S_{tj}\sum_{i=1}^{N-j}\frac{S^k_{ti}}{S_0}) - K, 0)$$

where will simulate the future stock prices at $t = j$ from $t = j + 1$ to $t = N$, which are $N - j$ points.

$$S_{ti} = S_0 e^{\sigma B_{ti} + (\mu - \frac{1}{2}\sigma^2)t}$$

And simulate n times, then take the average value of the asian call.

For the Asian call value computed by geometric average, we use the following close formula:

$$C^{A,g}_t = e^{-r(T-t)}(G_t^{\frac{t}{T}} S_t^{\frac{(T-t)}{T}} e^{\bar{\mu}+\bar{\sigma}^2} N(p_1) - KN(p_2))$$

where:

$$G_t = e^{\frac{1}{t}\int_0^t log(S_u)du}$$

$$p_2 = \frac{\frac{t}{T}log(G_t) + \frac{T-t}{T}log(S_t) + \bar{\mu} - log(K)}{\bar{\sigma}}$$

$$p1 = p2 + \bar{\sigma}$$

So at time 0, we get the Asian call value by Monte Carlo arithmetic average is 3.499. As for geometric average method , we compute the call value by the close formula and it is 3.325. The value of vanilla call is 8.021 with same underlying asset and K. And the vanilla call value is higher than Asian call.

1.3 Asian call value for whole path

We use the parameters of section 1.2 and the formulas for arithmetic average and geometric average to compute the Asian call value for whole stock path, which is shown as follows:

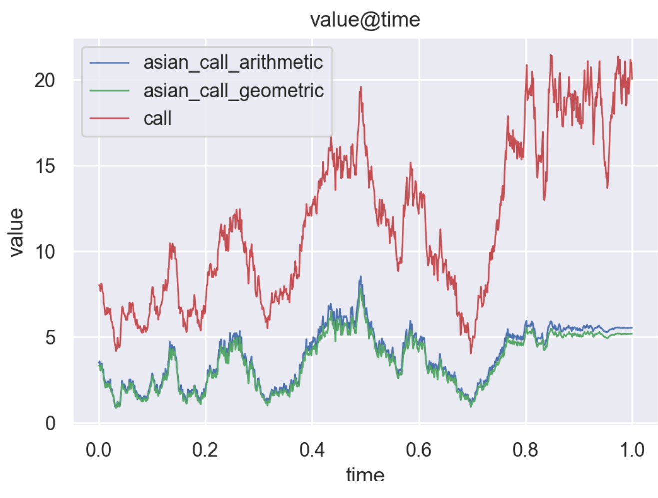

Figure 2: Asian call value

We can see that Asian call values computed by arithmetic average and geometric average are very close to each other. But the call value of geometric average is just a little lower just because the geometric average is slightly less than arithmetic average. At the same time, Asian options are less expensive than their standard counterparts, as the volatility of the average price is less than the volatility of the spot price.

2 Bond prices with Vasicek model for interest rates

2.1 Simulate several realizations

We simulate with different parameters for b, a, σ and $T = 1, N = 1000$. We can see mean-reverting phenomena from interest rate paths. For example, we take $a = 0.05$, and the figure 4 shows several interest rate path, we can see that the paths will get close to $a = 0.05$. And in addition, we get the distribution of the last value of the interest paths by simulating the path for 10000 times. Then we compute the average value of the last values. We get 0.0499, which is very close to a. At the same time , we can see from the picture that most of the last value of the paths are lying around $a = 0.05$. So in conclusion, we can say that there is a very obvious mean-reverting phenomena.

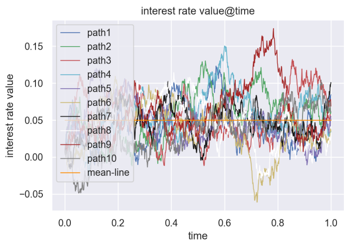

Figure 3: interest rate paths

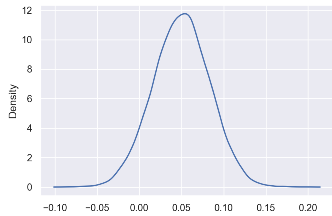

Figure 4: the distribution of last interest

2.2 T-bond price

For T-bond price, we compute by the following formulas:

$$p_t^T = e^{A_t^T - r_t B_t^T}$$

where:

$$A_t^T = \frac{1}{b^2}(B_t^T - T + t)(b^2 a - \frac{1}{2}\sigma^2) - \frac{\sigma^2 (B_t^T)^2}{4b}$$

$$B_t^T = \frac{1}{b}(1 - e^{-b(T-t)})$$

So we can plot T-bond prices for several interest paths, which is shown as following. All the bond price will end up at 1 at time maturity. Most of the trends are going up as time goes forward although there are some fluctuations in the bond prices. As we increase the value of b, the bond become less volatile.

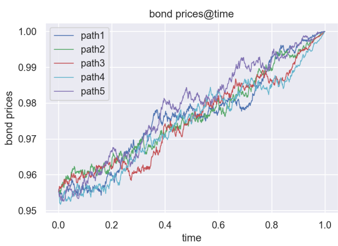

Figure 5: bond prices
$r_0 = 0.01, a = 0.05, b = 10$

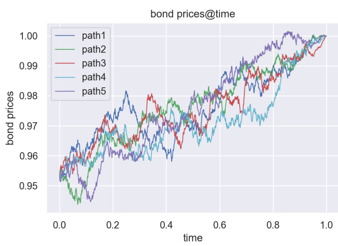

Figure 6: bond prices
$r_0 = 0.03, a = 0.05, b = 10$

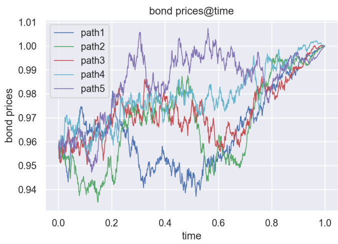

Figure 7: bond prices
$r_0 = 0.02, a = 0.05, b = 5$

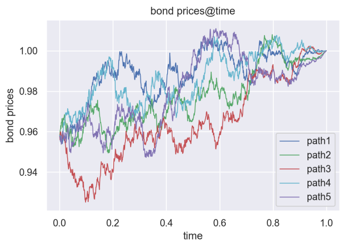

Figure 8: bond prices
$r_0 = 0.08, a = 0.05, b = 1$

2.3 Yield to maturity

To compute the yield to maturity, we use the formula:

$$y_t^T = -\frac{1}{T-t} log(p_t^T)$$

We compute the yield to maturity by change the time to maturity. At the same time, we will use different r, σ, a, b values. The plot is shown as following. As we can see from the plot, the yield to maturity is not always a concave function. It depends on the parameters. My finding is that, when the r_0 is less than a by a lot, the function is concave. But when r_0 is approaching a(here $a = 5\%$), the fact will change, it is not concave anymore.If r_0 is higher than a by a lot, then the function becomes convex. In addition, for all the paths, we find that as time to maturity increases, yield to maturity is getting closer to $a = 0.05$.

Possible explanation: r_0 is the short-run interest rate and we can regard a as long-term interest rate because of mean-reverting. So if the short-run interest rate is higher than long-run interest rate, the yield to maturity will decrease as the time to maturity increases and the function is convex. If the short-run interest rate is lower than long-run interest rate, the yield to maturity will increase as the time to maturity increases and the function is concave. If the r_0 is close to a, it is not very clear.

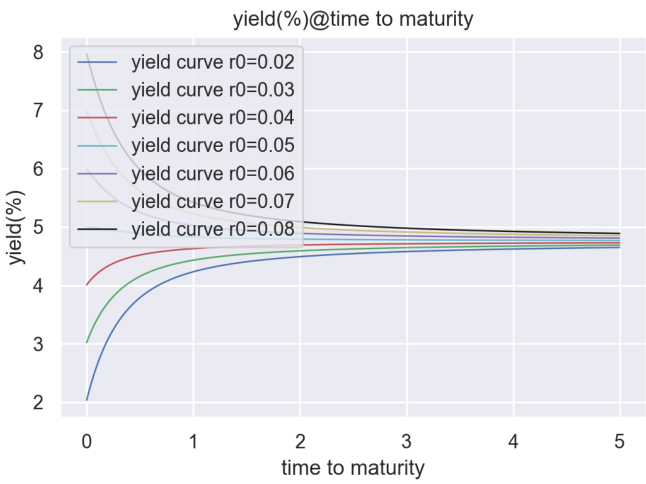

Figure 9: yield to maturity:$a = 0.05$

```python
import math
import os
import random
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib._color_data as mcd
import matplotlib.patches as mpatch
from scipy.stats import norm
sns.set()


# simulate stock prices paths:
def St_simulation_SDE(S0, T, N, r, sig):
    dt = T/N
    dBs = np.random.normal(0, dt**0.5, N)
    path = []
    St = S0
    path.append(St)
    for i,dB_i in enumerate(dBs):
        St = St + sig*St*dB_i+r*St*dt
        path.append(St)
    return path


# N(d1)
def d1_v(St, K, sigma, expir_t, r):
    d1 = (math.log(St/K) + (r+0.5*sigma**2)*expir_t)/(sigma*expir_t**0.5)
    N1 = norm.cdf(d1)
    return N1


# N(d2)
def d2_v(St, K, sigma, expir_t, r):
    d2 = (math.log(St/K) + (r-0.5*sigma**2)*expir_t)/(sigma*expir_t**0.5)
    N2 = norm.cdf(d2)
    return N2


# Call option vaue at time t
def Ct_v(St, K, sigma, expir_t, r):
    N1 = d1_v(St, K, sigma, expir_t, r)
    N2 = d2_v(St, K, sigma, expir_t, r)
    Ct = St*N1 - K*math.exp(-r*expir_t)*N2
    return Ct


# call option path
def Ct_value(path, K, T, N, sigma):
    dt = T/N
    Ct_path = []
    Deltas = []
    Deltas_dis = []
    for i in range(N):
        St = path[i]
        Ti = T - i*dt
        Ct = Ct_v(St, K, sigma, Ti, r)
        Ct_path.append(Ct)
    return Ct_path


# get the arithmetic average
def avg_arith_val(path):
    return np.sum(path)/len(path)


# get the geometric average
def avg_geome_val(path):
    return np.prod(np.array(path)/100)**(1/len(path))*100

```

```

# get arithmetic average path:
def avg_arith_path(path):
    arith_path = []
    for i in range(len(path)):
        avg_arith = avg_arith_val(path[0:i+1])
        arith_path.append(avg_arith)
    return arith_path


# get the geometric average path:
def avg_geome_path(path):
    geome_path = []
    for i in range(len(path)):
        avg_geome = avg_geome_val(path[0:i+1])
        geome_path.append(avg_geome)
    return geome_path


# Monte Carlo for asian call
def asian_monte_v(sum_now, St, S0, r, sig, K, T, N, step_i, simulate_times):
    dt = T/N
    N_left = N+1 - step_i
    et = T - (step_i-1)*dt
    disconut_f = math.exp(-r*et)
    Cts = []
    for i in range(simulate_times):
        if N_left>0:
            future_path = St_simulation_SDE(S0, et, N_left, r, sig)
            future_path = future_path[1:]
            all_sum = sum_now + St/S0 * np.sum(future_path)
            all_avg = all_sum/(N+1)
        else:
            all_avg = sum_now/(N+1)
        Ct = disconut_f* max(all_avg-K, 0)
        Cts.append(Ct)
    #print('the number future', len(future_path))
    simulate_Ct = np.sum(Cts)/len(Cts)
    return simulate_Ct


# Monte Carlo simulation for asian call path
def asian_monte_path(stock_path, r, sig, K, T, N, simulate_times):
    simulate_Cts = []
    length = len(stock_path)
    S0 = stock_path[0]
    for i in range(length):
        if i%100==0:
            print('processnig {} steps'.format(i))
        St = stock_path[i]
        sum_now = np.sum(stock_path[0:i+1])
        #print('the number until now', len(stock_path[0:i+1]))
        step_i = i+1
        simulate_Ct = asian_monte_v(sum_now, St, S0,
                      r, sig, K, T, N, step_i, simulate_times)
        simulate_Cts.append(simulate_Ct)
    return simulate_Cts


# pricing asian option with geometric average:
def Gt(gone_path, Nt):
    Gt_v = math.exp(1/Nt*np.sum(np.log(np.array(gone_path))))
    return Gt_v

def mu_bar(r, sig, T, et):
    mu_bar_v = (r-sig**2/2)*et**2/(2*T)
    return mu_bar_v

def sig_bar(sig, et):
    sig_bar2_v = sig**2*et**3/(3*T**2)
```

```
    return sig_bar2_v

def P2(t, et, St, Gt, mu_bar, sig_bar, K):
    p2_v = (t/T*math.log(Gt)+et/T*math.log(St)+mu_bar-math.log(K))/sig_bar
    return p2_v

def P1(t, et, St, Gt, mu_bar, sig_bar, K):
    p2_v = P2(t, et, St, Gt, mu_bar, sig_bar, K)
    p1_v = p2_v + sig_bar
    return p1_v


# asian option price at time t
def asian_gemom_v(r, sig, T, t, et, St, gone_path, step_i):
    Gt_v = Gt(gone_path, step_i)
    mu_bar_v = mu_bar(r, sig, T, et)
    sig_bar_v = sig_bar(sig, et)**0.5
    p1_v = P1(t, et, St, Gt_v, mu_bar_v, sig_bar_v, K)
    NP1 = norm.cdf(p1_v)
    p2_v = P2(t, et, St, Gt_v, mu_bar_v, sig_bar_v, K)
    NP2 = norm.cdf(p2_v)
    disconut_f = math.exp(-r*et)
    GSENP1 = Gt_v**(t/T)*St**(et/T)*math.exp(mu_bar_v+sig_bar_v**2/2)*NP1
    Ct = disconut_f*(GSENP1-K*NP2)
    return Ct


def asian_gemom(stock_path, r, sig, T, N, K):
    dt = T/N
    Cts = []
    for i in range(N):
        t = i*dt
        et = T - t
        gone_path = stock_path[0:i+1]
        St = stock_path[i]
        step_i = i + 1
        Ct = asian_gemom_v(r, sig, T, t, et, St, gone_path, step_i)
        Cts.append(Ct)
    return Cts


# exercise 2 ------------------------#
# simulate the interest rate path
def Rt_simulation_SDE(r0, T, N, a, b, sig):
    dt = T/N
    dBs = np.random.normal(0, dt**0.5, N)
    path = []
    rt = r0
    path.append(rt)
    for i,dB_i in enumerate(dBs):
        rt = rt + sig*dB_i+b*(a-rt)*dt
        path.append(rt)
    return path


def ATBT(a, b, T, t, sig):
    et = T - t
    BT_v = 1/b*(1-math.exp(-b*et))
    AT_v = 1/b**2*(BT_v - T + t)*(b**2*a - 0.5*sig**2)- sig**2*(BT_v)**2/(4*b)
    return AT_v, BT_v

def bond_price(rt, a, b, sig, T, t,  et):
    AT_v, BT_v = ATBT(a, b, T, t, sig)
    return math.exp(AT_v-rt*BT_v)

def bond_price_path(rts, a, b, sig, T, N):
    dt = T/N
    price_path = []
    for i in range(N+1):
        rt = rts[i]
```

```
        t = i*dt
        et = T - t
        price_i = bond_price(rt, a, b, sig, T, t, et)
        price_path.append(price_i)
    return price_path


def yield_curve(r0, Tb, a, b, sig):
    t = 0
    dt = Tb/N
    yTs = []
    for i in range(1000):
        T = (i+1)*dt
        et = T
        bond_price_i = bond_price(r0, a, b, sig, T, t, et)
        yT = -1/T * math.log(bond_price_i)
        yTs.append(yT)
    return yTs


# plot
def plot_path(paths, names, T, xname, yname, img_path=None):
    colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'w', 'brown', 'gray', 'darkorange']
    num = len(names)
    plt.figure(figsize=(6, 4))
    plt.rcParams['savefig.dpi'] = 200
    plt.rcParams['figure.dpi'] = 200
    for i in range(num):
        path_i = paths[i]
        num_ps = len(path_i)
        index = [i*T/(num_ps-1) for i in range(num_ps)]
        plt.plot(index, path_i, color=colors[i],
                 linestyle="-", linewidth=0.9, label=names[i])

    plt.xlabel(xname)
    plt.ylabel(yname)
    plt.title('{}@{}'.format(yname,xname))
    plt.legend()
    if img_path != None:
        plt.savefig(img_path)
    plt.show()


# step1: plot arithmetic and geometric average:
def plot_avgs(stock_path, T,  img_path=None):
    arith_path = avg_arith_path(stock_path)
    geome_path = avg_geome_path(stock_path)
    paths = [stock_path, arith_path, geome_path]
    names = ['spot value', 'arithmetic average', 'geometric average']
    xname = 'time'
    yname = 'value'
    plot_path(paths, names, T, xname, yname, img_path)


def exercise1():
    r = 0.05
    K = 105
    S0 = 100
    Sigma = 0.20
    N = 1000
    T = 1
    simulate_times = 3000
    stock_path = St_simulation_SDE(S0, T, N, r, Sigma)
    save_root = './'
    avgs_path_img = os.path.join(save_root, 'avgs.png')
    plot_avgs(stock_path, T, img_path=avgs_path_img)

    # pricing by monte carlo with arithmetic average
    Aaisn_Cts_arith = asian_monte_path(stock_path, r, Sigma, K, T, N, simulate_times)
    Aaisn_Cts_geome = asian_gemom(stock_path, r, Sigma, T, N, K)
```

```python
    Cts = Ct_value(stock_path, K, T, N, Sigma)
    print(Aaisn_Cts_arith[0], Aaisn_Cts_geome[0], Cts[0])
    calls_path = [Aaisn_Cts_arith, Aaisn_Cts_geome, Cts]
    names = ['asian_call_arithmetic', 'asian_call_geometric', 'call']
    xname = 'time'
    yname = 'value'
    calls_path_img = os.path.join(save_root, 'calls.png')
    plot_path(calls_path, names, T, xname, yname, img_path=calls_path_img)

## 2.1 Simulate several realizations
T = 1
N = 1000
sig = 0.15
r0 = 0.01
a = 0.05
b = 10
lasts = []
paths = []
names = []
j = 0
for i in range(10000):
    Rts = Rt_simulation_SDE(r0, T, N, a, b, sig)
    lasts.append(Rts[-1])
    if i % 1000 == 0:
        j += 1
        paths.append(Rts)
        names.append('path{}'.format(j))
mean = np.sum(lasts)/len(lasts)
mean_line = [mean for i in range(len(Rts))]
paths.append(mean_line)
names.append('mean-line')
print('the mean value of last point {}'.format(mean))
yname = 'interest rate value'
xname = 'time'
calls_path_img = './interest_paths.png'
plot_path(paths, names, T, xname, yname, img_path=calls_path_img)
sns_plot = sns.kdeplot(lasts, x="last value")
fig = sns_plot.get_figure()
fig.savefig("last_distribution.png")

## 2.2 bond prices
bond_path = []
for path_i in paths:
    price_path = bond_price_path(path_i, a, b, sig, T, N)
    bond_path.append(price_path)
yname = 'bond prices'
xname = 'time'
calls_path_img = './bond_prices.png'
plot_path(bond_path, names, T, xname, yname, img_path=calls_path_img)

## 2.3 yield to maturity
sig = 0.15
a,b = 0.05, 5
r0 = 0.01
yts = []
names = []
T = 5
for i in range(5):
    r0 = r0 + i* 0.01
    yt = yield_curve(r0, T, a, b, sig)
    yt = (np.array(yt)*100).tolist()
    yts.append(yt)
    names.append('yield curve r0={}'.format(round(r0,2)))

yname = 'yield(%)'
xname = 'time'
path_img = './yield.png'
plot_path(yts, names, T, xname, yname, img_path=path_img)
```