FIN 503: Advanced derivatives **(Due: 3/11/2020)**

Assignment #7

Professor: Elena Perazzi *Assistant:* Marc-Aurèle Divernois

Students: Hien Lê, Francesco Maizza, Anita Mezzetti

Problem 1

Pricing an Asian option with the LSMC method and constant volatility

In order to price our option it is required first of all to simulate the price paths over one year with dt= 1/4. In this way we a price path with constant volatility, time intervals equal to 3 months and maturity equal to 1 year. the resulting paths are shown in figure 1. It is important to notice that since the drift is negative the price paths will arrive on average after one year to a level lower than the original price.

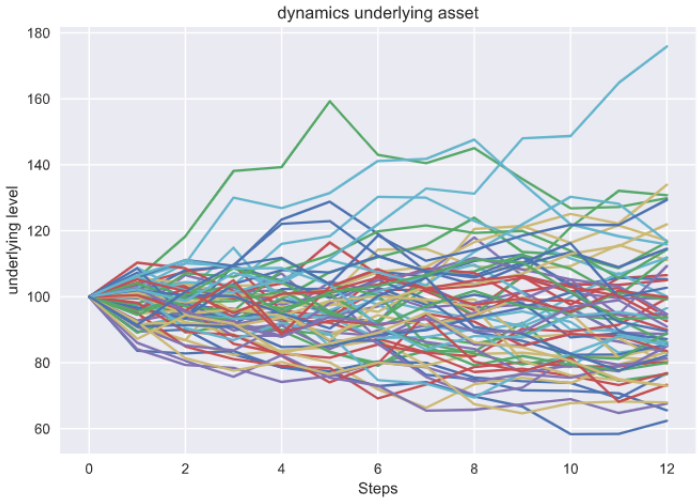


Figure 1: Price dynamics

The next step is to determine the payoff function at each exercise point. The payoff is determined by the difference between the average price of the underlying asset at the exercise dates (process A_t) and the constant strike. These are given by the question itself.

This is the main difference between the Asian option and Bermudan option, since in the latter the payoff depends only on the price at the underlying at the the execution price. One result of this is that the price of an European call is not a lower boundary for the price of an Asian option as in the case of the Bermudan option.

In order to run the least square Monte Carlo method we define the variables for in-the-money paths on which the regression is needed for determining the continuation value. They are the following: the underlying asset price and its powers of order 2 and 3 (S, S^2, S^3), and the process A (average of asset prices at any given exercise date, defined in the question) and its powers

of order 2 and 3 (A, A^2, A^3), these variables, along with a column of ones that represents the constant whose coefficient is the intercept, form the matrix X in the polynomial regression. To reduce the computational cost of the algorithm the beta is calculated as the OLS estimator $(X^T X)^{-1}(X^T Y)$, where Y is the sum of the discounted cash flows.

This algorithm resulted in the following price and standard deviation:

$$\text{price} = 7.0674 \quad \text{standard deviation} = 10.345\%$$

However, it is important to notice that from the distribution of the prices in figure 2 that on average it is much more likely that we never exercise the option than we exercise it. This is due to the fact that we have an underlying asset that has a negative drift. This high number of 0 values is also one of the reasons why we an relatively high standard error in this Monte Carlo simulation even with 100000 paths.

Figure 2: Distribution of the simulated prices

Pricing an Asian option with the LSMC method with implied volatility

The algorithm for deriving the price follows the same steps of the case with constant volatility, however, the difference here is in the method for simulating the underlying price path.

After having obtained the desired volatilises for our given K/S0 we can interpolate these volatility in order to have monthly values. With the latter values we can the simulate our price path with monthly intervals. In this case we have an average volatility 0.231975 so we can aspect a result similar to the result of a constat volatility.

After having determined the values of the underlying at 3m 6m 9m 12m the pricing follows as in the above point.

This algorithm will result in the following price and standard deviation:

$$\text{price} = 6.99295 \quad \text{standard deviation} = 10.569\%$$

Also in this case we can notice that from the distribution of the prices 3 that on average is much more likely that we never exercise the option than we exercise it

Figure 3: Distribution of the simulated prices

We can notice that the method that uses the implied volatility generates a slightly lower price of the case in which we have constant volatility. However, in order to infer meaningful conclusions about the two results we need to run a test to check if the two prices are significantly different from each other.
Indeed, by running a test of the sample means we can conclude that we cannot reject that price determined with constant volatility statistically is equal to the price determined with implied volatility.

Appendix: Python code

In [776]:
```
import math
import numpy as np
import pandas as pd
import scipy.stats as si
import seaborn as sns
```

In [787]:
```
### parameters
S0 = 100
K = 98
T = [0,0.25,0.5,0.75,1]
r = 0.0
delta = 0.02
mcpath = 100000
n = len(T)
np.random.seed(420)
```

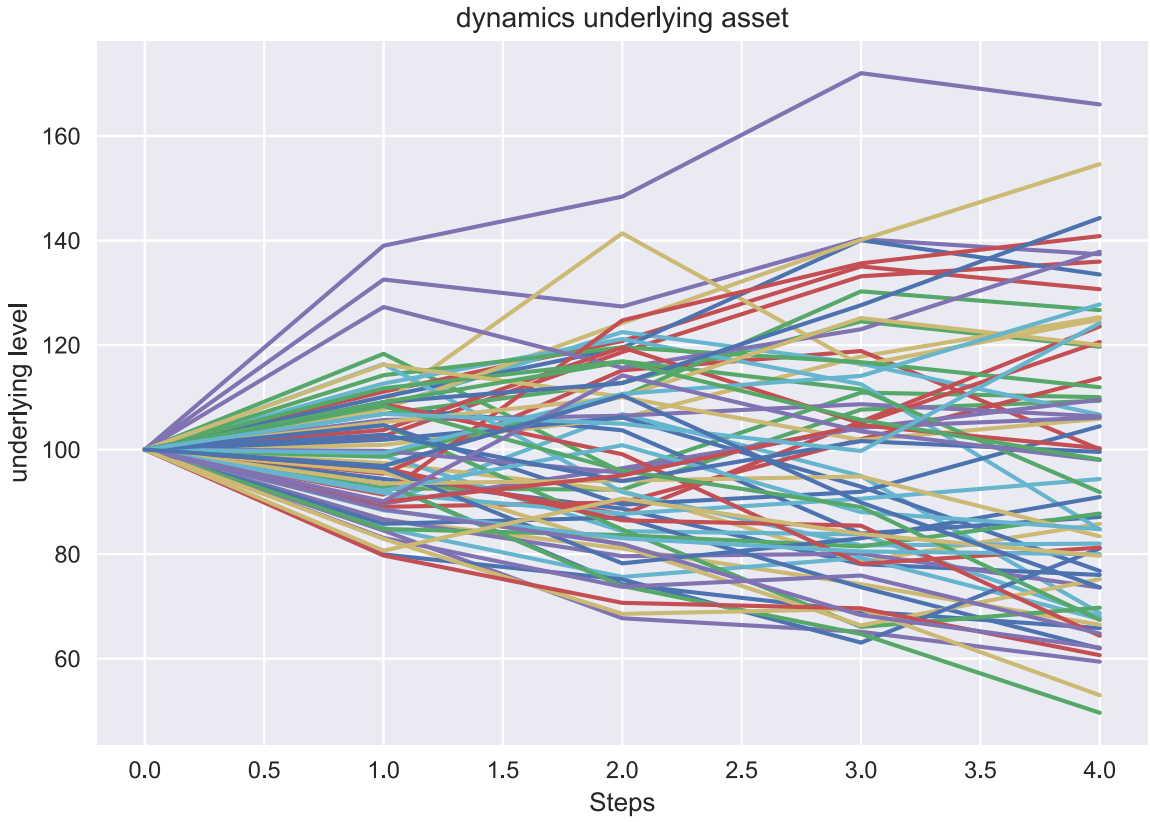
PART A

In [788]:
```
n = len(T) # number of periods
sigma = 0.23 # Constant sigma
dt = 1/4 # Constat time interval
S = np.zeros((mcpath,n))
for i in range(0,mcpath):
    t = 0
    W = 0
    C=np.ones(5)
    for j in range(0,n):
        if j==0:
            C[j]=100
        else:
            W = np.random.normal(0,np.sqrt(dt))
            C[j]=np.exp((r-delta-0.5*sigma**2)*dt+sigma*W)
        S[i,j] = np.prod(C) # price path
```

In [789]:
```
sdff=pd.DataFrame(S)
import matplotlib.pyplot as plt

plt.style.use('seaborn')

plt.plot(sdff.T.loc[:,0:1000:15])
plt.grid(True)
plt.title('dynamics underlying asset')
plt.xlabel('Steps')
plt.ylabel('underlying level')
plt.show()
```



In [790]:
```
# Compute A for t1 t2 t3 t4
A = np.zeros((mcpath,n))
A[:,1] = S[:,1]
A[:,2] = (S[:,1] + S[:,2])/2
A[:,3] = (S[:,1] + S[:,2] + S[:,3])/3
A[:,4] = (S[:,1] + S[:,2] + S[:,3] + S[:,4])/4
```

In [791]:
```
## define function for the intrinsic value
def intr(a):
    res = np.array([])
    for k in range(0,len(a)):
        if a[k] > 0:
            res = np.append(res,a[k])
        else:
            res = np.append(res,0)
    return res
```

In [792]:
```
C = np.zeros((mcpath,n))
C[:,n-1] = intr(A[:,n-1]-K) # intrinsic value at the last period
for i in range(1,n):
    X = np.array([np.ones(mcpath),S[:,n-1-i],S[:,n-1-i]**2,S[:,n
-1-i]**3,A[:,n-1-i],A[:,n-1-i]**2,A[:,n-1-i]**3]).T # variables
on which
    Y = np.sum(C,axis = 1) # cashflows received if not exercised
    beta = np.dot(np.linalg.pinv(np.dot(X.T,X)),np.dot(X.T,Y)) #
beta coefficients
    Y_reg = np.dot(X,beta) # continuation value
    stop = intr(A[:,n-1-i]-K) > Y_reg # determing optimal stoppi
ng points
    C[:,n-1-i] = intr(A[:,n-1-i]-K)*stop # assign optimal stoppi
ng values to the cash flows
    for j in range(0,mcpath):
        if (stop[j] == True):
            if (C[j,n-i] > 0):
                C[j,n-i:n] = 0 # delate cashflows after optimal
strike
p=np.amax(C, 1) # when is never optimal to exrcise before maturi
ty continuation value                          increasing overtim
e, hence optimal exercise at the last maturity
```

In [793]:
```
price=sum(p.T)/mcpath # price using LSMC with constant volatilit
y
price
```

Out[793]: 7.067443990946188

In [794]:
```
dff=pd.DataFrame(p.T)
dff.describe() # summary statistic of the LSMC
```

Out[794]:

	0
count	100000.000000
mean	7.067444
std	10.344693
min	0.000000
25%	0.000000
50%	0.362104
75%	11.714036
max	92.959051

In [814]:
```
ax=sns.distplot(dff[0], hist=True, kde=True,
             bins=int(180/5), color = 'blue',
             hist_kws={'edgecolor':'black'})
ax.set_title("Distribution of the simulated prices")
ax.set_ylabel('Probability')
ax.set_xlabel('price')
```

Out[814]: Text(0.5, 0, 'price')

In [796]:
```
## BSM price of an European Call which is the lower boundary of
the price of an american       call
def bs(S, K, T, r, q, sigma):

    d1 = (np.log(S / K) + (r - q + 0.5 * sigma ** 2) * T) / (sig
ma * np.sqrt(T))
    d2 = (np.log(S / K) + (r - q - 0.5 * sigma ** 2) * T) / (sig
ma * np.sqrt(T))

    call = (S * np.exp(-q * T) * si.norm.cdf(d1, 0.0, 1.0) - K *
np.exp(-r * T) * si.norm.cdf(d2, 0.0, 1.0))

    return call
bs(S0, K, 1,r,delta, sigma)
```

Out[796]: 8.983224650185882

PART B

In [827]:
```
## import implied volatilites from problem set 5
ind= pd.read_csv('./strikes.csv',names=['s'])/2772.70
mat=pd.read_csv('./maturities.csv')
v=list(mat.columns)
vol= pd.read_csv('./impv_total.csv',names=v)
vol.index=ind.s
```

In [798]:
```
localvol=vol.loc[K/S0] # get the implied volatiliets for our K/S
0
localvol=pd.concat([pd.Series([0]), localvol])
```

In [799]:
```
localvol=localvol.iloc[0:localvol.index.get_loc('1')+1] # consid
er volatility upto T=1
```

In [800]:
```
xp=[float(i) for i in localvol.index]
d=np.arange(0, 13, 1)/12
d=d.tolist()
localvolint=np.interp(d, xp, localvol) #interpolate local volati
lities so to have montlhy
periods
```

In [822]:
```
np.std(localvolint[1:])
```

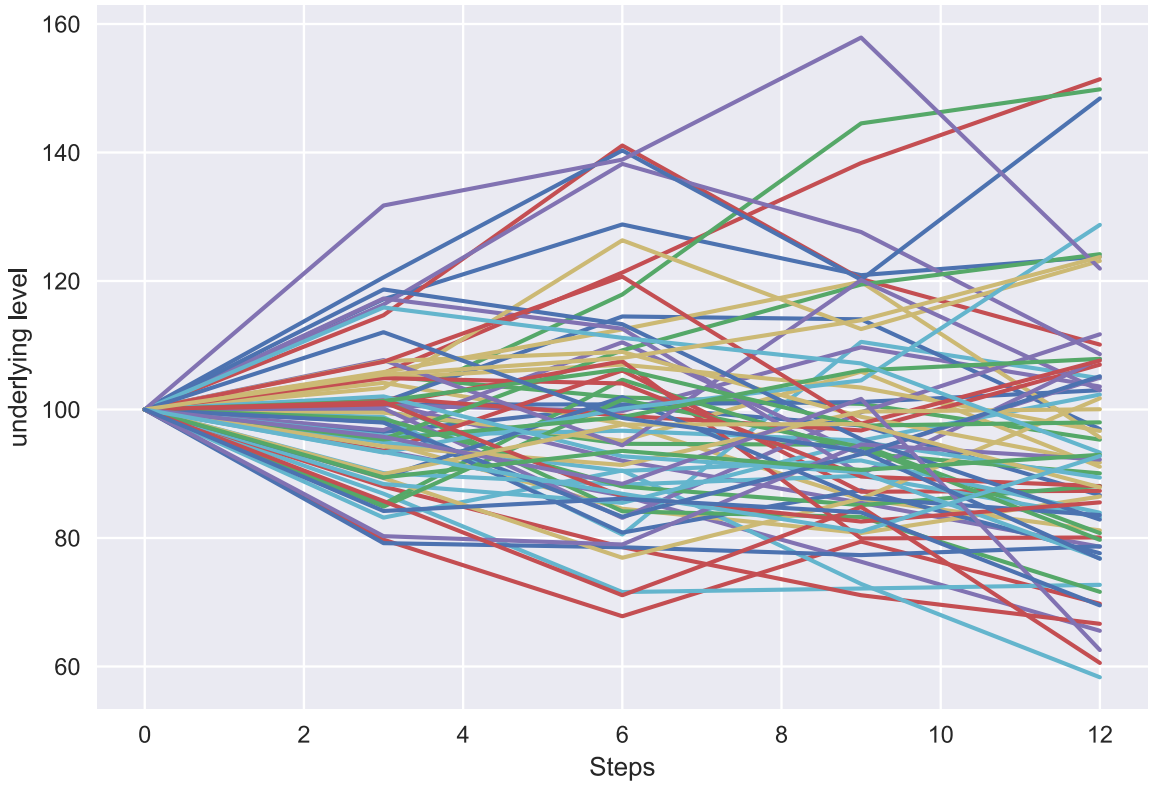
Out[822]: 0.0017893173924356854

In [801]:
```
nl=len(localvolint)
S = np.zeros((mcpath,nl))
for i in range(0,mcpath):
    t = 0
    W = 0
    C=np.ones(nl)
    j=0
    dtl=1/12
    for j in range(0,nl):
        if j==0:
            C[j]=100
        else:
            W = np.random.normal(0,np.sqrt(dtl))
            C[j]=np.exp((r-delta-0.5*localvolint[j]**2)*dtl+loca
lvolint[j]*W)
        S[i,j] = np.prod(C) # generate underlying paths
```

In [804]:
```
import matplotlib.pyplot as plt

plt.style.use('seaborn')


plt.plot(sdf.T.loc[:,0:1000:15])
plt.grid(True)
plt.xlabel('Steps')
plt.ylabel('underlying level')
plt.show()
```



In [802]:
```
sdf=pd.DataFrame(S)
sdf=sdf.loc[:,0::3]
Sl=sdf.to_numpy() # select the price at 3m 6m 9m 12m
```

In [803]:
```
# Compute A for t1 t2 t3 t4
Al = np.zeros((mcpath,n))
Al[:,1] = Sl[:,1]
Al[:,2] = (Sl[:,1] + Sl[:,2])/2
Al[:,3] = (Sl[:,1] + Sl[:,2] + Sl[:,3])/3
Al[:,4] = (Sl[:,1] + Sl[:,2] + Sl[:,3] + Sl[:,4])/4
```

In [805]:
```
p2=np.zeros((mcpath,1))
Cl = np.zeros((mcpath,n)) # always possible to exercise in 5 per
iods
Cl[:,n-1] = intr(Al[:,n-1]-K) # intrinsic value at the last peri
od
for i in range(1,5):
    X = np.array([np.ones(mcpath),Sl[:,n-1-i],Sl[:,n-1-i]**2,Sl[
:,n-1-i]**3,Al[:,n-1-i],Al[:,n-1-i]**2,Al[:,n-1-i]**3]).T # vari
ables on which regression for countinuation value is
run
    Y = np.sum(Cl,axis = 1) # future cashflows received if not e
xercised
    beta = np.dot(np.linalg.pinv(np.dot(X.T,X)),np.dot(X.T,Y)) #
beta coefficients
    Y_reg = np.dot(X,beta) # continuation value
    stop = intr(Al[:,n-1-i]-K) > Y_reg # determing optimal stopp
ing points
    Cl[:,n-1-i] = intr(Al[:,n-1-i]-K)*stop # assign optimal stop
ping values to the cashflows
    if (stop[j] == True):
            if (Cl[j,n-1-i] > 0):
                Cl[j,n-i:n] = 0 # delate cashflows after optimal
strike

p2=np.amax(Cl, 1)
```

In [806]:
```
price=sum(p2.T)/mcpath # price using LSMC with local volatility
price
```

Out[806]: 6.99295203173467

In [807]:
```
df=pd.DataFrame(p2.T)
df.describe()  # summary statistic of the LSMC
```

Out[807]:

	0
count	100000.000000
mean	6.992952
std	10.569438
min	0.000000
25%	0.000000
50%	0.000000
75%	11.492476
max	111.730339

In [815]:
```
ax=sns.distplot(df[0], hist=True, kde=True,
             bins=int(180/5), color = 'blue',
             hist_kws={'edgecolor':'black'})
ax.set_title("Distribution of the simulated prices")
ax.set_ylabel('Probability')
ax.set_xlabel('price')
```

Out[815]: Text(0.5, 0, 'price')

In [809]:
```
## test statistics on the difference of the two estimantions
from scipy import stats
t_check=stats.ttest_ind(p,p2,equal_var=False)
t_check
alpha=0.05
if(t_check[1]<alpha):
    print('price determined with constant volatility statistical
ly different from the price determined with implied volatility '
)
else:
   print(' we cannot reject that price determined with constant
volatility statistically is equal to the price determined with i
mplied volatility')
```

```
 we cannot reject that price determined with constant volatility
statistically is eqaul to the price determined with implied volat
ility
```

In [ ]: