Li_Jiaqi_Project5

February 22, 2019

1 Problem 1

The followings are the funcntions for computing option prices:

```
In [2]: import numpy as np
        import time
        import matplotlib.pyplot as plt

        #Stock Price
        def StockPrices(S0, r, sd, T, paths, steps):
            dt = T/steps
            # Generate stochastic process and its antithetic paths
            Z = np.random.normal(0, 1, paths//2 * (steps)).reshape(paths//2,(steps))
            Z_inv = -Z
            dWt = np.sqrt(dt) * Z
            dWt_inv = np.sqrt(dt) * Z_inv
            # bind the normal and antithetic Wt
            dWt = np.concatenate((dWt, dWt_inv), axis=0)
            # define the initial value of St
            St = np.zeros((paths, steps + 1))
            St[:, 0] = S0
            for i in range (steps):
                St[:, i+1] = St[:, i]*np.exp((r - 1/2*(sd**2))*dt + sd*dWt[:, i])
            return St[:,1:]

        # transform observations by different method for regression analysis
        def f_reg(x,k,method):
            x = np.array(x)
            n = len(x)
            if method == "Laguerre":
                if k == 1:
                    R1 = np.exp(-x/2)
                    return R1
                elif k == 2:
                    R1 = np.exp(-x/2)
                    R2 = np.exp(-x/2)*(1-x)
                    return np.array([R1,R2]).reshape(2,n)
```

```
    elif k == 3:
        R1 = np.exp(-x/2)
        R2 = np.exp(-x/2)*(1-x)
        R3 = np.exp(-x/2)*(1-2*x+x**2/2)
        return np.array([R1,R2,R3]).reshape(3,n)
    elif k == 4:
        R1 = np.exp(-x/2)
        R2 = np.exp(-x/2)*(1-x)
        R3 = np.exp(-x/2)*(1-2*x+x**2/2)
        R4 = np.exp(-x/2)*(1-3*x+3*x**2/2-x**3/6)
        return np.array([R1,R2,R3,R4]).reshape(4,n)
if method == "Hermite":
    if k == 1:
        R1 = [1]*n
        return R1
    elif k == 2:
        R1 = [1]*n
        R2 = 2*x
        return np.array([R1,R2]).reshape(2,n)
    elif k == 3:
        R1 = [1]*n
        R2 = 2*x
        R3 = 4*x**2-2
        return np.array([R1,R2,R3]).reshape(3,n)
    elif k == 4:
        R1 = [1]*n
        R2 = 2*x
        R3 = 4*x**2-2
        R4 = 8*x**3-12*x
        return np.array([R1,R2,R3,R4]).reshape(4,n)
if method == "Monomials":
    if k == 1:
        R1 = [1]*n
        return R1
    elif k == 2:
        R1 = [1]*n
        R2 = x
        return np.array([R1,R2]).reshape(2,n)
    elif k == 3:
        R1 = [1]*n
        R2 = x
        R3 = x**2
        return np.array([R1,R2,R3]).reshape(3,n)
    elif k == 4:
        R1 = [1]*n
        R2 = x
        R3 = x**2
        R4 = x**3
```

```
            return np.array([R1,R2,R3,R4]).reshape(4,n)

# LSMC process
def f_APLS(S0, r, sd, T, paths, steps, K, k, methods):
    dt = T/steps
    St = StockPrices(S0, r, sd, T, paths, steps)
    # initialize payoffs matrix
    payoffs = np.zeros((paths, steps))
    payoffs[:,steps - 1] = np.maximum(K - St[:,steps - 1], 0)
    # initialize stopping time matrix
    index = np.zeros((paths, steps))
    index[:,steps-1] = np.where(payoffs[:,steps - 1]> 0, 1, 0)
    discI = np.array(index[:,steps-1])
    Ot = np.array(np.maximum(K - St[:,steps-1],0))
    # initialize continuation value matrix
    for j in reversed(range(steps - 1)):
        payoffs[:,j] = np.maximum(K - St[:, j],0)
        # Find in the money paths
        In = np.where(Ot*np.exp(-r*dt*discI) > 0)[0]
        #  Use x which are in the money
        X = f_reg(St[In, j], k, methods)
        Y = np.array(Ot*np.exp(-r*dt*discI))[In]
        # Find Least Square Beta
        A = np.dot(X, X.T)
        b = np.dot(X, Y)
        Beta = np.dot(np.linalg.inv(A), b)
        # find full x
        x = f_reg(St[:, j], k, methods)
        # find continue value
        expected = np.dot(x.T, Beta)
        # update decision rule
        index[:, j] = np.where(np.array(payoffs[:,j]) - np.array(expected) > 0, 1, 0)
        for l in range(paths):
            if index[l,j] == 1:
                discI[l] = 1
                Ot[l] = K - St[l,j]
            elif index[l,j] == 0:
                discI[l] += 1
        payoffs[:,j] = np.maximum(np.array(payoffs[:,j]),np.maximum(np.array(expected)
    # Find the first occurence of 1, indicating the earlist exercise date
    first_exercise = np.argmax(index, axis = 1)
    index = np.zeros(shape = (paths, steps))
    index[np.arange(paths), first_exercise] = 1
    option = 0
    temp = index*payoffs
    for q in range(steps):
        option += np.mean(temp[:,q])*np.exp(-r*dt*(q+1))
    return option
```

(a)

```
In [5]: #parameters-----------------------------------------------------------
        N = 100000
        K = 40
        h = 100
        r = 0.06
        sigma = 0.2
        S01 = [36,40,44]
        T1 = [0.5,1,2]
        k1 = [2,3,4]

        #a-------------------------------------------------------------------
        method = "Laguerre"
        payoffa = np.zeros((3,9))
        payoffa[0,0] = f_APLS(S01[0],r,sigma,T1[0],N,h,K,k1[0],method)
        payoffa[0,1] = f_APLS(S01[0],r,sigma,T1[0],N,h,K,k1[1],method)
        payoffa[0,2] = f_APLS(S01[0],r,sigma,T1[0],N,h,K,k1[2],method)
        payoffa[0,3] = f_APLS(S01[0],r,sigma,T1[1],N,h,K,k1[0],method)
        payoffa[0,4] = f_APLS(S01[0],r,sigma,T1[1],N,h,K,k1[1],method)
        payoffa[0,5] = f_APLS(S01[0],r,sigma,T1[1],N,h,K,k1[2],method)
        payoffa[0,6] = f_APLS(S01[0],r,sigma,T1[2],N,h,K,k1[0],method)
        payoffa[0,7] = f_APLS(S01[0],r,sigma,T1[2],N,h,K,k1[1],method)
        payoffa[0,8] = f_APLS(S01[0],r,sigma,T1[2],N,h,K,k1[2],method)
        payoffa[1,0] = f_APLS(S01[1],r,sigma,T1[0],N,h,K,k1[0],method)
        payoffa[1,1] = f_APLS(S01[1],r,sigma,T1[0],N,h,K,k1[1],method)
        payoffa[1,2] = f_APLS(S01[1],r,sigma,T1[0],N,h,K,k1[2],method)
        payoffa[1,3] = f_APLS(S01[1],r,sigma,T1[1],N,h,K,k1[0],method)
        payoffa[1,4] = f_APLS(S01[1],r,sigma,T1[1],N,h,K,k1[1],method)
        payoffa[1,5] = f_APLS(S01[1],r,sigma,T1[1],N,h,K,k1[2],method)
        payoffa[1,6] = f_APLS(S01[1],r,sigma,T1[2],N,h,K,k1[0],method)
        payoffa[1,7] = f_APLS(S01[1],r,sigma,T1[2],N,h,K,k1[1],method)
        payoffa[1,8] = f_APLS(S01[1],r,sigma,T1[2],N,h,K,k1[2],method)
        payoffa[2,0] = f_APLS(S01[2],r,sigma,T1[0],N,h,K,k1[0],method)
        payoffa[2,1] = f_APLS(S01[2],r,sigma,T1[0],N,h,K,k1[1],method)
        payoffa[2,2] = f_APLS(S01[2],r,sigma,T1[0],N,h,K,k1[2],method)
        payoffa[2,3] = f_APLS(S01[2],r,sigma,T1[1],N,h,K,k1[0],method)
        payoffa[2,4] = f_APLS(S01[2],r,sigma,T1[1],N,h,K,k1[1],method)
        payoffa[2,5] = f_APLS(S01[2],r,sigma,T1[1],N,h,K,k1[2],method)
        payoffa[2,6] = f_APLS(S01[2],r,sigma,T1[2],N,h,K,k1[0],method)
        payoffa[2,7] = f_APLS(S01[2],r,sigma,T1[2],N,h,K,k1[1],method)
        payoffa[2,8] = f_APLS(S01[2],r,sigma,T1[2],N,h,K,k1[2],method)
```

```
In [10]: print(payoffa)
```

```
[[3.98984683 4.30434664 4.9985786  3.98661513 4.23667902 4.83082988
  3.96153952 4.23086954 4.71839743]
 [1.96955464 2.99310625 3.82229646 1.87156779 2.73579001 3.66376282
```

```
 1.90578616 2.46097305 3.24169838]
[1.66357342 2.54707725 2.90616366 1.25788503 2.09192763 3.31652232
 1.20093864 1.89875351 2.67496592]]
```

For k = 2:

T	$S_0 = 36$	$S_0 = 40$	$S_0 = 44$
0.5	3.98984683	1.96955464	1.66357342
1	3.98661513	1.87156779	1.25788503
2	3.96153952	1.90578616	1.20093864

For k = 3:

T	$S_0 = 36$	$S_0 = 40$	$S_0 = 44$
0.5	4.30434664	2.99310625	2.54707725
1	4.23667902	2.73579001	2.09192763
2	4.23086954	2.46097305	1.89875351

For k = 3:

T	$S_0 = 36$	$S_0 = 40$	$S_0 = 44$
0.5	4.9985786	3.82229646	2.90616366
1	4.83082988	3.66376282	3.31652232
2	4.71839743	3.24169838	2.67496592

```
In [6]: #b-----------------------------------------------------------------
        method = "Hermite"
        payoffb = np.zeros((3,9))
        payoffb[0,0] = f_APLS(S01[0],r,sigma,T1[0],N,h,K,k1[0],method)
        payoffb[0,1] = f_APLS(S01[0],r,sigma,T1[0],N,h,K,k1[1],method)
        payoffb[0,2] = f_APLS(S01[0],r,sigma,T1[0],N,h,K,k1[2],method)
        payoffb[0,3] = f_APLS(S01[0],r,sigma,T1[1],N,h,K,k1[0],method)
        payoffb[0,4] = f_APLS(S01[0],r,sigma,T1[1],N,h,K,k1[1],method)
        payoffb[0,5] = f_APLS(S01[0],r,sigma,T1[1],N,h,K,k1[2],method)
        payoffb[0,6] = f_APLS(S01[0],r,sigma,T1[2],N,h,K,k1[0],method)
        payoffb[0,7] = f_APLS(S01[0],r,sigma,T1[2],N,h,K,k1[1],method)
        payoffb[0,8] = f_APLS(S01[0],r,sigma,T1[2],N,h,K,k1[2],method)
        payoffb[1,0] = f_APLS(S01[1],r,sigma,T1[0],N,h,K,k1[0],method)
        payoffb[1,1] = f_APLS(S01[1],r,sigma,T1[0],N,h,K,k1[1],method)
        payoffb[1,2] = f_APLS(S01[1],r,sigma,T1[0],N,h,K,k1[2],method)
        payoffb[1,3] = f_APLS(S01[1],r,sigma,T1[1],N,h,K,k1[0],method)
        payoffb[1,4] = f_APLS(S01[1],r,sigma,T1[1],N,h,K,k1[1],method)
        payoffb[1,5] = f_APLS(S01[1],r,sigma,T1[1],N,h,K,k1[2],method)
        payoffb[1,6] = f_APLS(S01[1],r,sigma,T1[2],N,h,K,k1[0],method)
        payoffb[1,7] = f_APLS(S01[1],r,sigma,T1[2],N,h,K,k1[1],method)
```

```
        payoffb[1,8] = f_APLS(S01[1],r,sigma,T1[2],N,h,K,k1[2],method)
        payoffb[2,0] = f_APLS(S01[2],r,sigma,T1[0],N,h,K,k1[0],method)
        payoffb[2,1] = f_APLS(S01[2],r,sigma,T1[0],N,h,K,k1[1],method)
        payoffb[2,2] = f_APLS(S01[2],r,sigma,T1[0],N,h,K,k1[2],method)
        payoffb[2,3] = f_APLS(S01[2],r,sigma,T1[1],N,h,K,k1[0],method)
        payoffb[2,4] = f_APLS(S01[2],r,sigma,T1[1],N,h,K,k1[1],method)
        payoffb[2,5] = f_APLS(S01[2],r,sigma,T1[1],N,h,K,k1[2],method)
        payoffb[2,6] = f_APLS(S01[2],r,sigma,T1[2],N,h,K,k1[0],method)
        payoffb[2,7] = f_APLS(S01[2],r,sigma,T1[2],N,h,K,k1[1],method)
        payoffb[2,8] = f_APLS(S01[2],r,sigma,T1[2],N,h,K,k1[2],method)
```

In [13]:
```
print(payoffb)
```

```
[[3.980762   4.10075766 4.15766773 4.10732123 4.29638188 4.43558009
  4.24705543 4.48631844 5.13107888]
 [1.72094838 1.81807641 1.99834709 2.1557835  2.31625856 2.81996506
  2.56926664 2.78082952 4.13395728]
 [0.60708462 0.66285321 1.41209611 1.04518904 1.14928606 2.40493268
  1.54096439 1.69541551 3.59346935]]
```

For k = 2:

T	$S_0 = 36$	$S_0 = 40$	$S_0 = 44$
0.5	3.980762	1.72094838	0.60708462
1	4.10732123	2.1557835	1.04518904
2	4.24705543	2.56926664	1.54096439

For k = 3:

T	$S_0 = 36$	$S_0 = 40$	$S_0 = 44$
0.5	4.10075766	1.81807641	0.66285321
1	4.29638188	2.31625856	1.14928606
2	4.48631844	2.78082952	1.69541551

For k = 3:

T	$S_0 = 36$	$S_0 = 40$	$S_0 = 44$
0.5	4.15766773	1.99834709	1.41209611
1	4.43558009	2.81996506	2.40493268
2	5.13107888	4.13395728	3.59346935

In [7]:
```
        #c-------------------------------------------------------------------
        method = "Monomials"
        payoffc = np.zeros((3,9))
        payoffc[0,0] = f_APLS(S01[0],r,sigma,T1[0],N,h,K,k1[0],method)
```

```
        payoffc[0,1] = f_APLS(S01[0],r,sigma,T1[0],N,h,K,k1[1],method)
        payoffc[0,2] = f_APLS(S01[0],r,sigma,T1[0],N,h,K,k1[2],method)
        payoffc[0,3] = f_APLS(S01[0],r,sigma,T1[1],N,h,K,k1[0],method)
        payoffc[0,4] = f_APLS(S01[0],r,sigma,T1[1],N,h,K,k1[1],method)
        payoffc[0,5] = f_APLS(S01[0],r,sigma,T1[1],N,h,K,k1[2],method)
        payoffc[0,6] = f_APLS(S01[0],r,sigma,T1[2],N,h,K,k1[0],method)
        payoffc[0,7] = f_APLS(S01[0],r,sigma,T1[2],N,h,K,k1[1],method)
        payoffc[0,8] = f_APLS(S01[0],r,sigma,T1[2],N,h,K,k1[2],method)
        payoffc[1,0] = f_APLS(S01[1],r,sigma,T1[0],N,h,K,k1[0],method)
        payoffc[1,1] = f_APLS(S01[1],r,sigma,T1[0],N,h,K,k1[1],method)
        payoffc[1,2] = f_APLS(S01[1],r,sigma,T1[0],N,h,K,k1[2],method)
        payoffc[1,3] = f_APLS(S01[1],r,sigma,T1[1],N,h,K,k1[0],method)
        payoffc[1,4] = f_APLS(S01[1],r,sigma,T1[1],N,h,K,k1[1],method)
        payoffc[1,5] = f_APLS(S01[1],r,sigma,T1[1],N,h,K,k1[2],method)
        payoffc[1,6] = f_APLS(S01[1],r,sigma,T1[2],N,h,K,k1[0],method)
        payoffc[1,7] = f_APLS(S01[1],r,sigma,T1[2],N,h,K,k1[1],method)
        payoffc[1,8] = f_APLS(S01[1],r,sigma,T1[2],N,h,K,k1[2],method)
        payoffc[2,0] = f_APLS(S01[2],r,sigma,T1[0],N,h,K,k1[0],method)
        payoffc[2,1] = f_APLS(S01[2],r,sigma,T1[0],N,h,K,k1[1],method)
        payoffc[2,2] = f_APLS(S01[2],r,sigma,T1[0],N,h,K,k1[2],method)
        payoffc[2,3] = f_APLS(S01[2],r,sigma,T1[1],N,h,K,k1[0],method)
        payoffc[2,4] = f_APLS(S01[2],r,sigma,T1[1],N,h,K,k1[1],method)
        payoffc[2,5] = f_APLS(S01[2],r,sigma,T1[1],N,h,K,k1[2],method)
        payoffc[2,6] = f_APLS(S01[2],r,sigma,T1[2],N,h,K,k1[0],method)
        payoffc[2,7] = f_APLS(S01[2],r,sigma,T1[2],N,h,K,k1[1],method)
        payoffc[2,8] = f_APLS(S01[2],r,sigma,T1[2],N,h,K,k1[2],method)
```

```
In [12]: print(payoffc)
```

```
[[3.96578168 4.09240584 4.15935206 4.12399713 4.28892259 4.44138998
  4.23741356 4.49377137 5.18067749]
 [1.71752772 1.81176717 2.00787536 2.15425966 2.31539083 2.79966248
  2.59333676 2.78306822 4.11608838]
 [0.60564376 0.66071687 1.46076053 1.05258392 1.15187396 2.22237022
  1.53129959 1.69719182 3.517246  ]]
```

For k = 2:

T	$S_0 = 36$	$S_0 = 40$	$S_0 = 44$
0.5	3.96578168	1.71752772	0.60564376
1	4.12399713	2.15425966	1.05258392
2	4.23741356	2.59333676	1.53129959

For k = 3:

T	$S_0 = 36$	$S_0 = 40$	$S_0 = 44$
0.5	4.09240584	1.81176717	0.66071687
1	4.28892259	2.31539083	1.15187396
2	4.49377137	2.78306822	1.69719182

For k = 3:

T	$S_0 = 36$	$S_0 = 40$	$S_0 = 44$
0.5	4.15935206	2.00787536	1.46076053
1	4.44138998	2.79966248	2.22237022
2	5.18067749	4.11608838	3.517246

(d)

By using all three methods of Monomials, Hermite and Laguerre, the American put option at T=0.5 with S_0 = 36, 40, 44 are approximately **4.2, 1.9, 0.6**; prices at T=1 are **4.5, 2.3, 1.1**; prices at T=2 are **5.0, 2.8, 1.7**. Also, we find that method of Laguerre causes larger estimated errors while the other two methods (Monomials and Hermite) are quite consistent and stable.

2 Problem 2

(a)

```
In [8]: #a-----------------------------------------------------------------
        def f_FSEP(S0,r,sigma,t,T,paths,steps,method):
            dt = T/steps
            price = StockPrices(S0,r,sigma,T,paths,steps)
            K = price[:,int(0.2/dt)]
            option = np.mean(np.maximum(K-price[:,steps-1],0))*np.exp(-r)
            return option

        EuroP_FS = f_FSEP(65.0,0.06,0.2,0.2,1,100000,100,"Monomials")

        print(EuroP_FS)
```

```
3.136013240938645
```

Forward-start European put option price is $3.136

(b)

```
In [9]: #b-----------------------------------------------------------------
        def f_FSAP(S0,r,sd,k,t,T,paths,steps,methods):
            dt = T/steps
            St = StockPrices(S0, r, sd, T, paths, steps)
            stop = int(0.2/dt)
            K = St[:,stop]
```

```
        # initialize payoffs matrix
        payoffs = np.zeros((paths, steps))
        payoffs[:,steps - 1] = np.maximum(K - St[:,steps - 1], 0)
        # initialize stopping time matrix
        index = np.zeros((paths, steps))
        index[:,steps-1] = np.where(payoffs[:,steps - 1]> 0, 1, 0)
        discI = np.array(index[:,steps-1])
        Ot = np.array(np.maximum(K - St[:,steps-1],0))
        # initialize continuation value matrix
        for j in reversed(range(stop,steps - 1)):
            payoffs[:,j] = np.maximum(K - St[:, j],0)
            # Find in the money paths
            In = np.where(Ot*np.exp(-r*dt*discI) > 0)[0]
            #  Use x which are in the money
            X = f_reg(St[In, j], k, methods)
            Y = np.array(Ot*np.exp(-r*dt*discI))[In]
            # Find Least Square Beta
            A = np.dot(X, X.T)
            b = np.dot(X, Y)
            Beta = np.dot(np.linalg.inv(A), b)
            # find full x
            x = f_reg(St[:, j], k, methods)
            # find continue value
            expected = np.dot(x.T, Beta)
            # update decision rule
            index[:, j] = np.where(np.array(payoffs[:,j]) - np.array(expected) > 0, 1, 0)
            for l in range(paths):
                if index[l,j] == 1:
                    discI[l] = 1
                    Ot[l] = K[l] - St[l,j]
                elif index[l,j] == 0:
                    discI[l] += 1
            payoffs[:,j] = np.maximum(np.array(payoffs[:,j]),np.maximum(np.array(expected)
        # Find the first occurence of 1, indicating the earlist exercise date
        first_exercise = np.argmax(index, axis = 1)
        index = np.zeros(shape = (paths, steps))
        index[np.arange(paths), first_exercise] = 1
        option = 0
        temp = index*payoffs
        for q in range(steps):
            option += np.mean(temp[:,q]*np.exp(-r*dt*(q+1)))
        return option

    AmeriP_FS = f_FSAP(65,0.06,0.2,2,0.2,1,100000,100,"Monomials")

    print(AmeriP_FS)
```

```
3.327973794168421
```

Forward-start American put option price is $3.328