Quantitative Finance Homework 7

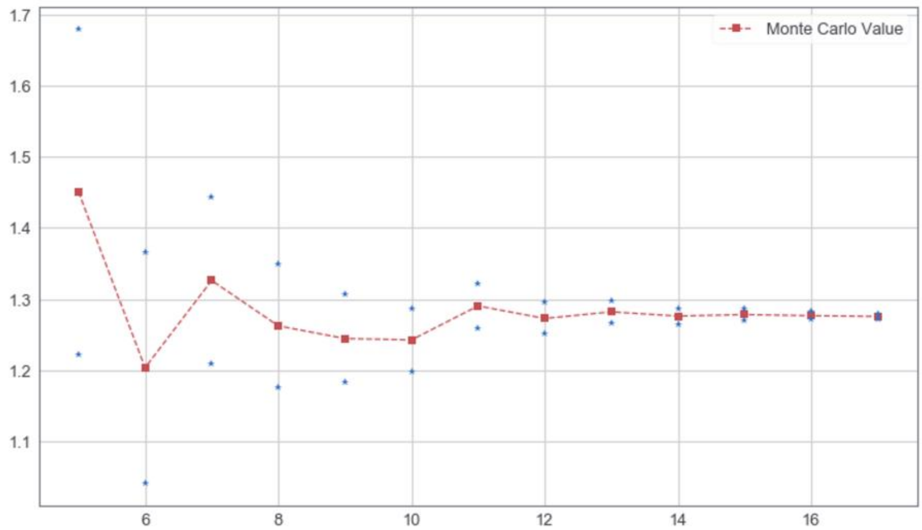
20121229 JunPyo Park

Setting Parameters

In [48]:
```
Rate = 0.05
Price = 5
Strike = 6
Time = 1
sigma = 0.3
L = 12 # number of time_step interval
time_line = np.linspace(0,Time,L)
dt = (Time) / L # unit size of time_step interval
```

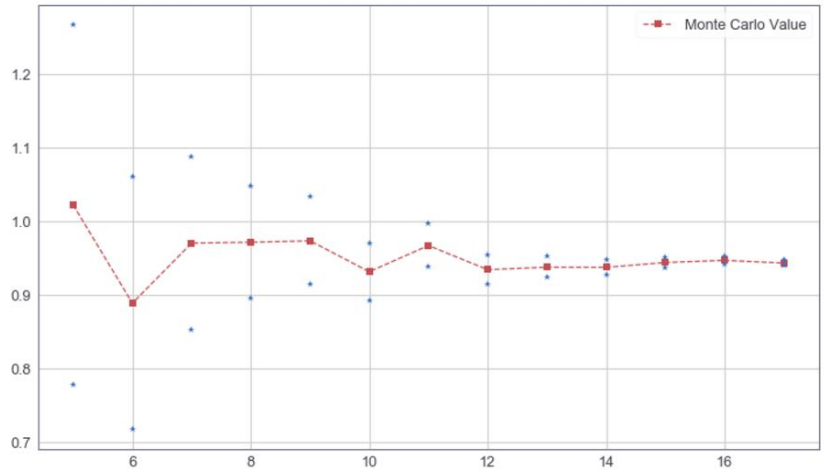
Arithmetic Average Asian Put Option

In [36]:
```
M = 2 ** (np.array(range(13)) + 5)
Exp = np.array(range(13)) + 5

V_ave = []
V_se = []

for i in M:

    paths=pd.DataFrame(np.ones(i)*Price)

    for j in range(L-1): # M(i) 개의 path 동시에 생성
        paths[j+1] = paths[j].apply(lambda x : x* np.exp((Rate - 0.5 * sigma * sigma ) * dt + sigma * np.sqrt(dt) * np.random.randn()))


    final_value = (paths.sum(axis=1) - Price) / L # using arithmetic mean
    V = final_value.apply(lambda x : max(np.exp(-Rate * Time) * (Strike-x),0)) # M(i) 개의 final_value 들에 대한 put value 계산


    V_ave.append(V.mean())
    V_se.append(V.std()/np.sqrt(i))


V_ave = np.array(V_ave)
V_se = np.array(V_se)

plt.figure(figsize=(12,7))
plt.plot(Exp, V_ave, '--rs', label = 'Monte Carlo Value')
plt.plot(Exp, V_ave + 1.96*V_se, '*b' )
plt.plot(Exp, V_ave - 1.96*V_se, '*b')
plt.legend()
plt.show()
```



Geometric Average Asian Put Option

In [46]:
```
M = 2 ** (np.array(range(13)) + 5)
Exp = np.array(range(13)) + 5

V_ave = []
V_se = []

for i in M:

    paths=pd.DataFrame(np.ones(i)*Price)

    for j in range(L-1): # M(i) 개의 path 동시에 생성
        paths[j+1] = paths[j].apply(lambda x : x* np.exp((Rate - 0.5 * sigma * sigma ) * dt + sigma * np.sqrt(dt) * np.random.randn()))


    final_value = (paths.prod(axis=1)-Price) ** (1/L) # using geometric mean
    V = final_value.apply(lambda x : max(np.exp(-Rate * Time) * (Strike-x),0)) # M(i) 개의 final_value 들에 대한 put value 계산


    V_ave.append(V.mean())
    V_se.append(V.std()/np.sqrt(i))


V_ave = np.array(V_ave)
V_se = np.array(V_se)

plt.figure(figsize=(12,7))
plt.plot(Exp, V_ave, '--rs', label = 'Monte Carlo Value')
plt.plot(Exp, V_ave + 1.96*V_se, '*b' )
plt.plot(Exp, V_ave - 1.96*V_se, '*b')
plt.legend()
plt.show()
```



Up-and-In barrier call

In [47]:
```
B=15
```

- An up-and-out call option formula is

$$S\left(N(d_1)-N(e_1)-\left(\frac{B}{S}\right)^{1+2r/\sigma^2}(N(f_2)-N(g_2))\right)$$
$$-Ee^{-r(T-t)}\left(N(d_2)-N(e_2)-\left(\frac{B}{S}\right)^{-1+2r/\sigma^2}(N(f_1)-N(g_1))\right).$$

where d_1 and d_2 are defined before and $e_1, e_2, f_1, f_2, g_1, g_2$ are

$$e_1=\frac{\log(S/B)+(r+\frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}},$$
$$e_2=\frac{\log(S/B)+(r-\frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}},$$
$$f_1=\frac{\log(S/B)-(r-\frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}},$$
$$f_2=\frac{\log(S/B)-(r+\frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}},$$
$$g_1=\frac{\log(SE/B^2)-(r-\frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}},$$
$$g_2=\frac{\log(SE/B^2)-(r+\frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}}.$$

10

In [100]:
```
d_1 = (np.log(Price/Strike) + (Rate + 0.5 * sigma * sigma)*Time)/(sigma*np.sqrt(Time))
d_2 = d1 - sigma * np.sqrt(Time)
```

In [87]:
```
e_1 = (np.log(Price/B) + (Rate+0.5*sigma*sigma)*Time) / sigma * np.sqrt(Time)
e_1
```

Out[87]: -3.345374295560366

In [88]:
```
e_2 = (np.log(Price/B) + (Rate-0.5*sigma*sigma)*Time) / sigma * np.sqrt(Time)
e_2
```

Out[88]: -3.645374295560366

In [90]:
```
f_1 = (np.log(Price/B) - (Rate-0.5*sigma*sigma)*Time) / sigma * np.sqrt(Time)
f_1
```

Out[90]: -3.6787076288937

In [91]:
```
f_2 = (np.log(Price/B) - (Rate+0.5*sigma*sigma)*Time) / sigma * np.sqrt(Time)
f_2
```

Out[91]: -3.9787076288936993

In [92]:
```
g_1 = (np.log(Price * Strike/B**2) - (Rate-0.5*sigma*sigma)*Time) / sigma * np.sqrt(Time)
g_1
```

Out[92]: -6.733010068474216

In [93]:
```
g_2 = (np.log(Price * Strike/B**2) - (Rate+0.5*sigma*sigma)*Time) / sigma * np.sqrt(Time)
g_2
```

Out[93]: -7.0330100684742165

- An up-and-out call option formula is

$$S\left(N(d_1) - N(e_1) - \left(\frac{B}{S}\right)^{1+2r/\sigma^2}(N(f_2) - N(g_2))\right)$$
$$-Ee^{-r(T-t)}\left(N(d_2) - N(e_2) - \left(\frac{B}{S}\right)^{-1+2r/\sigma^2}(N(f_1) - N(g_1))\right).$$

In [102]:
```
a = Price*(norm.cdf(d_1) - norm.cdf(e_1) - (B/Price)**(1+2*Rate / sigma**2) * (norm.cdf(f_2) - norm.cdf(g_2)))
a
```

Out[102]: 1.9236751632129976

In [103]:
```
b = Strike * np.exp(-Rate * Time) * (norm.cdf(d_2) - norm.cdf(e_2) - (B/Price)**(-1+2*Rate / sigma**2) * (norm.cdf(f_1) - norm.cdf(g_1)))
b
```

Out[103]: 1.5807732800775711

In [105]:
```
call_price = bs_call_put(Rate, Price, Strike, Time, sigma)[0] # Calculated B-S Call Option Value
```

In [106]:
```
up_and_out_call = a - b
up_and_in_call = call_price - up_and_out_call
```

In [107]:
```
up_and_in_call
```

Out[107]: 0.0022979944115117945

In [117]:
```
def get_value(series): # One Path 를 입력받아 up-and-in 을 판단하여 value 를 return 하는 함수
    if series.max() > B:
        return max(np.exp(-Rate * Time) * (list(series)[-1]-Strike),0)
    else:
        return 0
```

In [116]:
```
M = 10 ** (np.array(range(4)) + 2)
Exp = np.array(range(4)) + 2
time_steps = [1e-2,1e-3,1e-4]

for dt in time_steps:
    V_ave = []
    V_se = []
    L = int(Time / dt) # number of time_step interval

    for i in M:

        paths=pd.DataFrame(np.ones(i)*Price)

        for j in range(L-1): # M(i) 개의 path 동시에 생성
            paths[j+1] = paths[j].apply(lambda x : x* np.exp((Rate - 0.5 * sigma * sigma ) * dt + sigma * np.sqrt(dt) * np.random.randn()))


        V = paths.apply(get_value, axis=1) # get_value 함수를 통해 M(i) 개의 value 계산


        V_ave.append(V.mean())
        V_se.append(V.std()/np.sqrt(i))


    V_ave = np.array(V_ave)
    V_se = np.array(V_se)

    plt.figure(figsize=(12,7))
    plt.plot(Exp, V_ave, '--rs', label = 'Monte Carlo Value')
    plt.plot(Exp, V_ave + 1.96*V_se, '*b' )
    plt.plot(Exp, V_ave - 1.96*V_se, '*b')
    plt.plot(Exp, up_and_in_call * np.ones(4))
    plt.title('dt = ' + str(dt))
    plt.legend()
    plt.show()
```