Homework 4

[redacted]

Problem 1

In [32]:

```
import numpy as np
import scipy.stats as stats
import math
```

In [3]:

```
np.random.seed(1)
m = 55000
d = 5000
MeanTBA = 1.0 # average interarrival time
MeanST = 0.8 # average service time

Y=0
Wait_times=[]
for i in range(0,d,1):
    A = np.random.exponential(MeanTBA, 1)
    X = np.sum(np.random.exponential(MeanST/3,3))
    Y = max(0, Y + X - A)
for i in range(d,m,1):
    A = np.random.exponential(MeanTBA, 1)
    X = np.sum(np.random.exponential(MeanST/3,3))
    Y = max(0, Y + X - A)
    Wait_times.append(Y)
```

In [22]:

```
# Use batching method to split the whole long replication into 20 batchings.
batching = []
for i in range(0, len(Wait_times), 2500):
    batching.append(Wait_times[i:i+2500])
```

In [26]:

```
for i in range(0,20,1):
    batching[i].sort()
```

In [55]:

```
# After sorting the data, we obtain a CI using the normal approximation for the
 Binomial formula。
#u = math.ceil(2500*0.8+1.96*np.sqrt(2500*0.8*0.2))
#l = math.floor(2500*0.8-1.96*np.sqrt(2500*0.8*0.2))
```

In [56]:

```
quantile = []
for j in range(0,20,1):
    quantile.append(batching[j][1999])
```

In [62]:

```
var_q = np.var(quantile)
quantile_80 = np.mean(quantile)
UCI = quantile_80 + stats.t.ppf(0.975, 20-1)*np.sqrt(var_q)/np.sqrt(20)
LCI = quantile_80 - stats.t.ppf(0.975, 20-1)*np.sqrt(var_q)/np.sqrt(20)
```

In [63]:

```
print("95% confidence interval for 0.8 quantile of the steady-state waiting time
is", (round(LCI,2),round(UCI,2)), "with 0.8 quantile is", round(quantile_80,2))
```

```
95% confidence interval for 0.8 quantile of the steady-state waiting
time is (3.46, 3.89) with 0.8 quantile is 3.68
```

Problem 2

In [3]:

```
import numpy as np
import math
import pandas as pd
import scipy.stats as stats
```

100 replications

In [6]:

```
np.random.seed(1)
Y_aver=[]
Y_vari=[]
N=100
X_hat = [[0.5,1,1,1,1], [1,0.5,1,1,1], [1,1,0.5,1,1], [0.3,1,1,0.4,1], [1,1,1,1,
0.5]]
for s in range(0,5,1):
    Y_list=[]
    for rep in range(0,N,1):
        X=[]
        for i in range(0,5,1):
            X.append(np.random.exponential(X_hat[s][i]))
        Y = max(X[0] + X[3], X[0] + X[2] + X[4], X[1] + X[4])
        Y_list.append(Y)
    Y_aver.append(np.mean(Y_list))
    Y_vari.append(np.var(Y_list))
```

In [8]:

```
SAN=pd.DataFrame({"Y_bar": Y_aver, "S_square": Y_vari})
SAN=SAN.T
SAN.columns = ["X1", "X2", "X3", "X4", "X5"]
SAN
```

Out[8]:

	X1	X2	X3	X4	X5
Y_bar	3.185770	3.111166	3.187353	2.921784	2.584679
S_square	1.874958	3.193679	1.913893	2.438957	1.588762

X5 has the smallest sample average.

The procedure with confidence level 0.95 returns the subset I = {X4, X5}.

In [17]:

```
alpha = 0.95 **(1/4)
# So the critical value of each scentrio is t alpha,99
```

In [34]:

```
Y_eva = []
for i in range(0,3,1):
    Y_eval = Y_aver[4]+(5.145*Y_vari[i]/100+5.145*Y_vari[4]/100)**0.5
    Y_eva.append(Y_eval)
Y_eva
```

Out[34]:

```
[3.0068266868359155, 3.0807202325864056, 3.0091926732103516]
```

Because the expected values of scenario 1, scenario 2 and scentrio 3 are more than 3.007, 3.081, 3.187 respectively, X1, X2 and X3 should be eliminated.

200 replications

In [4]:

```
np.random.seed(1)
Y_aver_200=[]
Y_vari_200=[]
N=200
X_hat = [[0.5,1,1,1,1], [1,0.5,1,1,1], [1,1,0.5,1,1], [0.3,1,1,0.4,1], [1,1,1,1,
0.5]]
for s in range(0,5,1):
    Y_list_200=[]
    for rep in range(0,N,1):
        X=[]
        for i in range(0,5,1):
            X.append(np.random.exponential(X_hat[s][i]))
        Y = max(X[0] + X[3], X[0] + X[2] + X[4], X[1] + X[4])
        Y_list_200.append(Y)
    Y_aver_200.append(np.mean(Y_list_200))
    Y_vari_200.append(np.var(Y_list_200))
```

In [5]:

```
SAN_200=pd.DataFrame({"Y_bar": Y_aver_200, "S_squared": Y_vari_200})
SAN_200=SAN_200.T
SAN_200.columns = ["X1", "X2", "X3", "X4", "X5"]
SAN_200
```

Out[5]:

	X1	X2	X3	X4	X5
Y_bar	3.019134	3.439222	2.849700	2.711548	2.969881
S_squared	1.958427	2.915197	2.116842	1.875488	2.245278

X4 has the smallest sample mean

In [6]:

```
ti_squared = (stats.t.ppf(0.987, 199))**2
Y_eva_200 = []
for i in [0,1,2,4]:
    Y_eval_200 = Y_aver_200[3]+(ti_squared*Y_vari_200[i]/100+ti_squared*Y_vari_2
00[3]/100)**0.5
    Y_eva_200.append(Y_eval_200)
Y_eva_200
```

Out[6]:

```
[3.150734503210934, 3.202485835127617, 3.1597161066252593, 3.1668679
58739491]
```

Because the expected values of scenario 2 is more than 3.202, X2 should be eliminated.

Problem 3

In [1]:

```
import numpy as np
from scipy import stats
import pandas as pd
```

In [31]:

```
np.random.seed(1)
Maturity = 1.0
InterestRate = 0.05
Sigma = 0.3
InitialValue = 50.0
StrikePrice = 55.0
Steps = 32
Interval = Maturity / Steps
Sigma2 = Sigma * Sigma / 2
Replications = 10000
Interval = Maturity / Steps
delta = [0.1, 0.5, 1, 5, 10]

ValueList_delta = [[] for i in range(5)]
ValueList = []
for rep in range(0,Replications):
    Sum = 0.0
    Sum_delta = []
    X_delta = []
    X = InitialValue
    for i in range(0,5,1):
        X_delta.append(InitialValue + delta[i])
        Sum_delta.append(0.0)
    for j in range(0,Steps):
        Z = np.random.standard_normal(1) # Use common random numbers.
        X = X * np.exp((InterestRate - Sigma2) * Interval + Sigma * np.sqrt(Inte
rval) * Z)
        Sum = Sum + X
        for d in range(0,5,1):
            X_delta[d] = X_delta[d] * np.exp((InterestRate - Sigma2) * Interval
+ Sigma * np.sqrt(Interval) * Z)
            Sum_delta[d] = Sum_delta[d] + X_delta[d]

    Value = np.exp(-InterestRate * Maturity) * max(Sum/Steps - StrikePrice, 0)
    ValueList.append(float(Value))
    for d in range(0,5,1):
        Value_delta = np.exp(-InterestRate * Maturity) * max(Sum_delta[d]/Steps
- StrikePrice, 0)
        ValueList_delta[d].append(float(Value_delta))
```

In [34]:

```
mean=[]
std=[]
FD_list = [[] for i in range(5)]
for i in range(0,5,1):
    for x,y in zip(ValueList_delta[i], ValueList):
        FD_list[i].append((x-y)/delta[i])
    mean.append(np.mean(FD_list[i]))
    std.append(np.std(FD_list[i]))
```

In [36]:

```
t_stat = stats.t.ppf(1-0.025, 10000-1)
UCI = []
LCI = []
Half_width = []
for x, y in zip(mean, std):
    UCI.append(x + (t_stat*y)/np.sqrt(10000))
    LCI.append(x - (t_stat*y)/np.sqrt(10000))
for i in range(0,5,1):
    Half_width.append(UCI[i]-LCI[i])
```

In [37]:

```
df = pd.DataFrame()
df['Delta'] = delta
df['Mean'] = mean
df['Confindent Interval'] = [(round(x,6),round(y,6)) for x,y in zip(UCI,LCI)]
df['Half-Width'] = Half_width
print(df.to_string(index=False))
```

```
 Delta      Mean   Confindent Interval  Half-Width
   0.1  0.377227  (0.388093, 0.366361)    0.021732
   0.5  0.385751  (0.396607, 0.374895)    0.021713
   1.0  0.395603   (0.406447, 0.38476)    0.021686
   5.0  0.474852  (0.485428, 0.464275)    0.021154
  10.0  0.569208  (0.579115, 0.559301)    0.019814
```

From the output, we can see with the drcreasing of delta, the estimates are converging to about 0.3X but at the meantime, the range of CI is increasing.

Problem 4

In [14]:

```
import scipy.stats as stats
import numpy as np
import pandas as pd
```

In [57]:

```
Maturity = 1.0
InterestRate = 0.05
Sigma = 0.3
InitialValue = 50.0
StrikePrice = 55.0
Sigma2 = Sigma * Sigma / 2
steps_list = [8,16,32,64,128]
Replications = 10000

Beta1_hat = []
Beta0_hat = []
Sigma_hat11 = []
Y_mean = []
Y_var = []
```

In [54]:

```
np.random.seed(1)

for S in steps_list:
    Ylist = []
    Clist = []
    Interval = Maturity / S
    T_new = (S+1)*Interval/2
    sigma_bar2 = (2*S+1)*Sigma**2/(3*S)
    sigma_bar = np.sqrt(sigma_bar2)
    delta = (Sigma**2-sigma_bar2)/2
    d = (np.log(InitialValue/StrikePrice)+(InterestRate-delta+sigma_bar2/2)*T_ne
w)/(sigma_bar*np.sqrt(T_new))
    Vc = np.exp(-delta*T_new)*InitialValue * stats.norm.cdf(d)-np.exp(-InterestR
ate*T_new)*StrikePrice*stats.norm.cdf(d-sigma_bar*np.sqrt(T_new))
    Vc = float(Vc)
    for i in range(0,Replications):
        Sum = 0.0
        Mul = 1
        X = InitialValue
        for j in range(0,S):
            Z = np.random.standard_normal(1)
            X = X * np.exp((InterestRate - Sigma2) * Interval + Sigma * np.sqrt(
Interval) * Z)
            Sum = Sum + X
            Mul = Mul * X
        Value1 = np.exp(-InterestRate * Maturity) * max(Sum/S - StrikePrice, 0)
        Value2 = np.exp(-InterestRate * Maturity) * max(Mul**(1/S)- StrikePrice,
0)
        Ylist.append(Value1)
        Clist.append(Value2)
    meanY = np.mean(Ylist)
    meanC = np.mean(Clist)

    Sum1 = 0
    Sum2 = 0
    Sum3 = 0
    for i in range(0,Replications):
        Sum1 += (Ylist[i]-meanY)*(Clist[i]-meanC)
        Sum2 += (Clist[i]-meanC)**2
    beta1 = float(Sum1/Sum2)
    beta0 = float(meanY-beta1*(meanC-Vc))
    Beta1_hat.append(beta1)
    Beta0_hat.append(beta0)

    for j in range(0,Replications):
        Sum3 += (Ylist[j]-beta0-beta1*(Clist[j]-Vc))**2
    Sigma_hat11.append(float(Sum3/(Replications - 2)*(1/Replications+(meanC-Vc)*
*2/Sum2)))
print(Beta0_hat)
print(Sigma_hat11)
```

```
[2.4410883639595817, 2.2652814041804095, 2.175257974109503, 2.128824
1360245284, 2.106326758194892]
[6.793041571049805e-06, 6.262495728186666e-06, 5.515731136487037e-0
6, 5.558677172475161e-06, 5.147943239411542e-06]
```

In [32]:

```
t_stat = stats.t.ppf(1-0.025, 10000-2)
UCI = []
LCI = []
Half_width = []
for x, y in zip(Beta0_hat, Sigma_hat11):
    LCI.append(x - t_stat*np.sqrt(y))
    UCI.append(x + t_stat*np.sqrt(y))
    Half_width.append(2*t_stat*np.sqrt(y))
```

In [43]:

```
df = pd.DataFrame()
df['Steps'] = steps_list
df['Beta0_hat'] = Beta0_hat
df['Confindent Interval'] = [(round(x,6),round(y,6)) for x,y in zip(LCI,UCI)]
df['Half-Width'] = Half_width
print(df.to_string(index=False))
```

```
 Steps  Beta0_hat   Confindent Interval  Half-Width
     8   2.441088  (2.435979, 2.446197)    0.010218
    16   2.265281  (2.260376, 2.270187)    0.009811
    32   2.175258  (2.170654, 2.179862)    0.009207
    64   2.128824  (2.124203, 2.133446)    0.009243
   128   2.106327  (2.101879, 2.110774)    0.008895
```

In conclusion, as the number of the steps increase, the Beta0_hat and the hald-widths of Beta0_hat are decreasing.