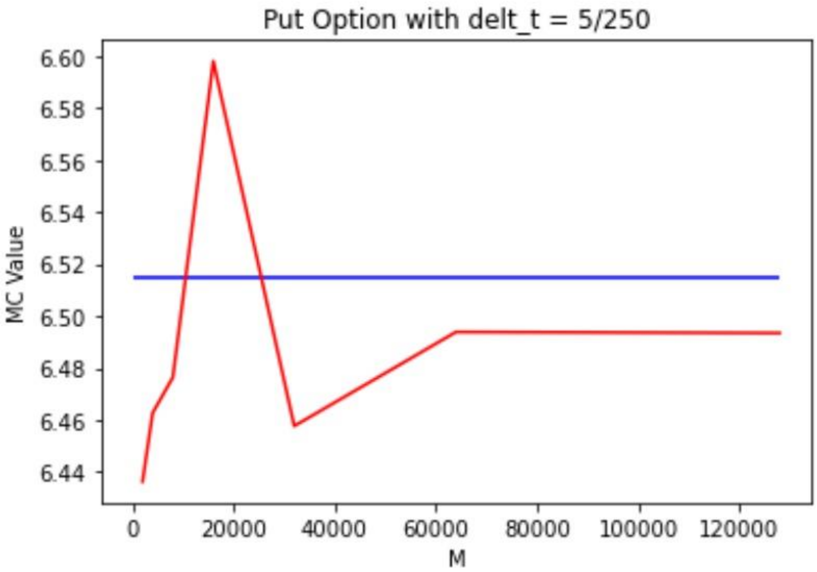
Sample Assignments of Python coding for financial applications:

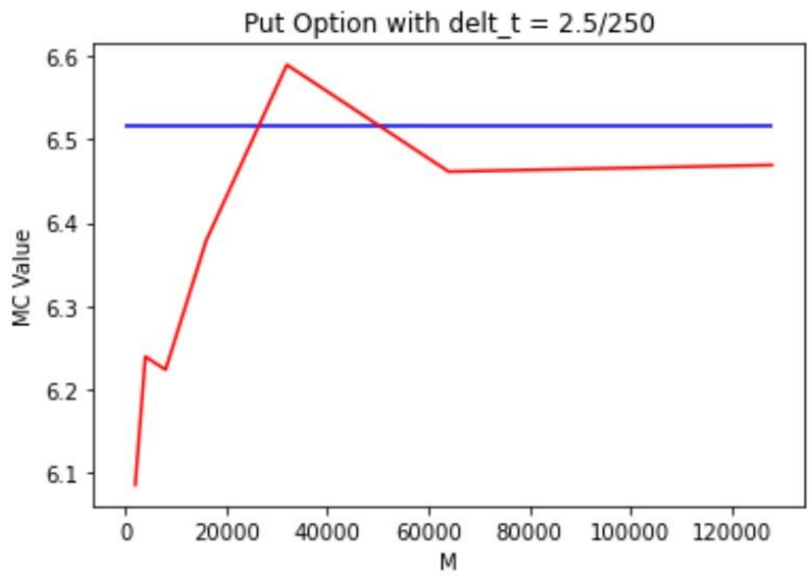
Black Scholes options/stocks pricing, Monte Carlo simulations for pricing, Delta hedging stochastic paths, CPPI simulations, etc.

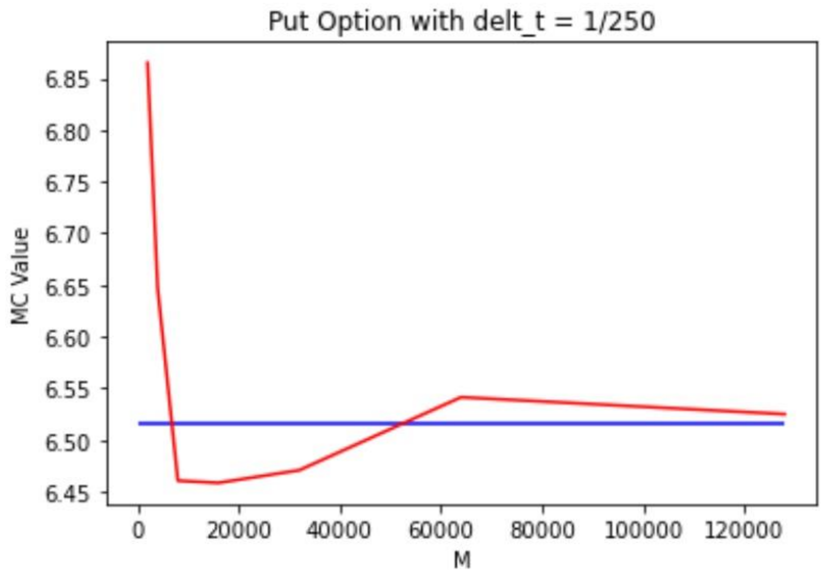
[1]:
```
## A4Q1
from scipy.stats import norm
import numpy as np
import math
import matplotlib.pyplot as plt
from astropy.table import Table

#adapted from Octave's financial toolkit
def blsprice(Price, Strike, Rate, Time, Volatility):
    sigma_sqrtT = Volatility * np.sqrt (Time)

    d1 = 1 / sigma_sqrtT * (np.log(Price / Strike) + (Rate + Volatility**2 / 2)␣
↪* Time)
    d2 = d1 - sigma_sqrtT

    phi1 = norm.cdf(d1)
    phi2 = norm.cdf(d2)
    disc = np.exp (-Rate * Time)
    F    = Price * np.exp ((Rate) * Time)

    Call = disc * (F * phi1 - Strike * phi2)
    Put  = disc * (Strike * (1 - phi2) + F * (phi1 - 1))
    return Call, Put

sigma=0.22
r=0.06
T=1.5
K=100
S0=100

def MC_put(delt_t,M):
    N=T/delt_t
    S_last=np.ones(M)*S0
    S_next=np.zeros(M)
    i=0
    while i < N:
```

```
        S_next=np.maximum(np.zeros(M), S_last+S_last*(r*delt_t+sigma*np.random.
↪normal(0,1,M)*math.sqrt(delt_t)))
        S_last=S_next
        i=i+1
    payoff=np.maximum(np.zeros(M), K-S_next)
    V=np.exp(-r*T)*np.sum(payoff)/M
    std=math.sqrt(np.sum((np.exp(-r*T)*payoff-V)**2)/(M-1))
    return V, std

M=[2000,4000,8000,16000,32000,64000,128000]
bls_put=blsprice(100,100,0.06,1.5,0.22)[1]


MC_list1=[]
j=0
while j <= 6:
    V=MC_put(5/250, M[j])[0]
    MC_list1.append(V)
    j=j+1


MC_list2=[]
k=0
while k <= 6:
    V=MC_put(2.5/250, M[k])[0]
    MC_list2.append(V)
    k=k+1


MC_list3=[]
u=0
while u <= 6:
    V=MC_put(1/250, M[u])[0]
    MC_list3.append(V)
    u=u+1


## delt_t = 5/250
plt.plot(M,MC_list1,'r')
plt.hlines(y=bls_put,xmin=100,xmax=128000,colors='blue')
plt.title("Put Option with delt_t = 5/250")
plt.ylabel('MC Value')
plt.xlabel('M')
#plt.legend(handles=[MC_line, exact_bls])
plt.show()
```

```
## delt_t = 2.5/250
plt.plot(M,MC_list2,'r')
plt.hlines(y=bls_put,xmin=100,xmax=128000,colors='blue')
plt.title("Put Option with delt_t = 2.5/250")
plt.ylabel('MC Value')
plt.xlabel('M')
plt.show()

## delt_t = 1/250
plt.plot(M,MC_list3,'r')
plt.hlines(y=bls_put,xmin=100,xmax=128000,colors='blue')
plt.title("Put Option with delt_t = 1/250")
plt.ylabel('MC Value')
plt.xlabel('M')
plt.show()
```




Put Option with delt_t = 2.5/250
MC Value
M
6.6
6.5
6.4
6.3
6.2
6.1
0
20000
40000
60000
80000
100000
120000


Put Option with delt_t = 1/250
MC Value
M
6.85
6.80
6.75
6.70
6.65
6.60
6.55
6.50
6.45
0
20000
40000
60000
80000
100000
120000

Comments: From the plots, the Monte Carlo values roughly approches the blue line (the exact Black-Scholes price) as the number of stimulations increase and vary around it generally.

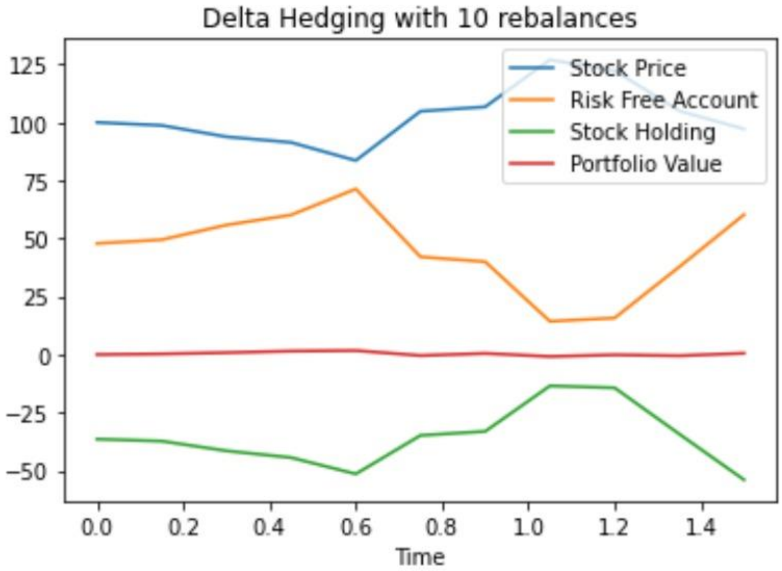
[4]:
```
MC_list4=[]
lower_list=[]
upper_list=[]
i=0
while i <= 6:
    V=MC_put(1/250,M[i])[0]
    std=MC_put(1/250,M[i])[1]
    lower_list.append(round(V-2.58*std/math.sqrt(M[i]),6))
    upper_list.append(round(V+2.58*std/math.sqrt(M[i]),6))
    MC_list4.append(round(V,6))
    i=i+1

## draw the 99% confidence interval table:
table_confidence = Table([M,MC_list4,lower_list,upper_list],
                      names=('M','Estimated Option Value',
                             'Lower Bound','Upper Bound'))
print('Put Option Table of 99% Confidence Interval with delt_t = 1/250')
print(table_confidence)
```

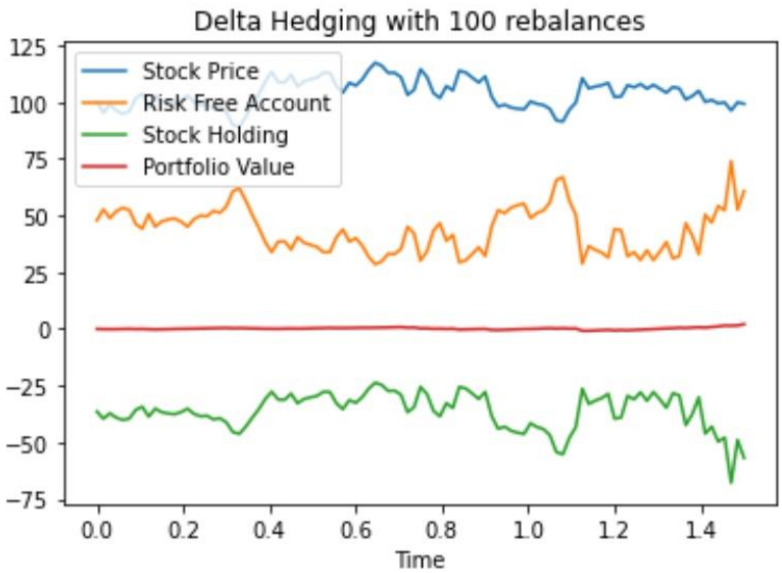
```
Put Option Table of 99% Confidence Interval with delt_t = 1/250
  M    Estimated Option Value Lower Bound Upper Bound
------ ---------------------- ----------- -----------
  2000                6.49725    5.891677    7.102823
  4000               6.343227    5.920164     6.76629
  8000               6.718594    6.420076    7.017112
 16000               6.377842    6.167301    6.588383
 32000               6.620691    6.472388    6.768993
 64000               6.537365    6.432024    6.642707
128000               6.533914    6.459185    6.608644
```

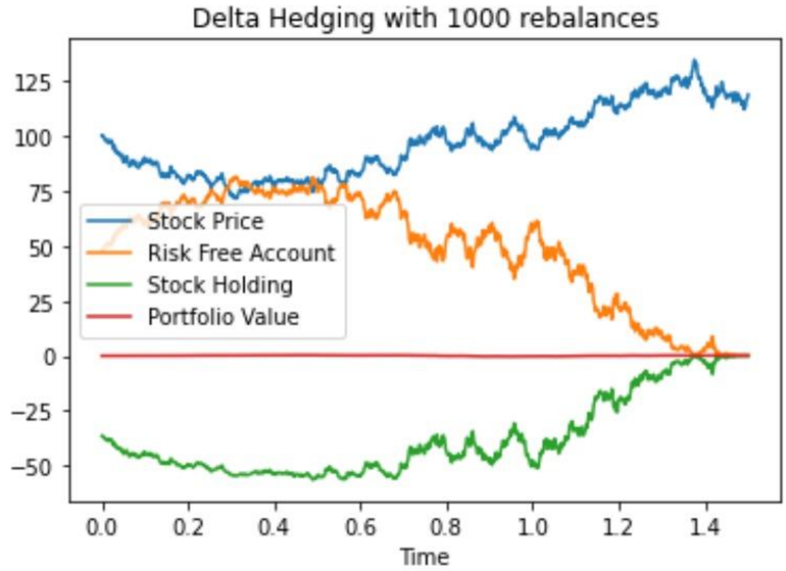
[22]:
```python
## A4Q2
from scipy.stats import norm
import numpy as np
import matplotlib.pyplot as plt
import math

#adapted from Octave's financial toolkit
def blsprice(Price, Strike, Rate, Time, Volatility):
    sigma_sqrtT = Volatility * np.sqrt (Time)

    d1 = 1 / sigma_sqrtT * (np.log(Price / Strike) + (Rate + Volatility**2 / 2)␣
 ↪* Time)
    d2 = d1 - sigma_sqrtT

    phi1 = norm.cdf(d1)
    phi2 = norm.cdf(d2)
    disc = np.exp (-Rate * Time)
    F    = Price * np.exp ((Rate) * Time)

    Call = disc * (F * phi1 - Strike * phi2)
    Put  = disc * (Strike * (1 - phi2) + F * (phi1 - 1))
    return Call, Put

#adapted from Octave's financial toolkit
def blsdelta(Price, Strike, Rate, Time, Volatility):
    d1 = 1 / (Volatility * np.sqrt(Time)) * (np.log (Price / Strike) + (Rate +␣
 ↪Volatility**2 / 2) * Time)

    phi = norm.cdf(d1)

    CallDelta = phi
    PutDelta = phi - 1
    return CallDelta, PutDelta

sigma = 0.3
r = 0.04
T = 1.5
```

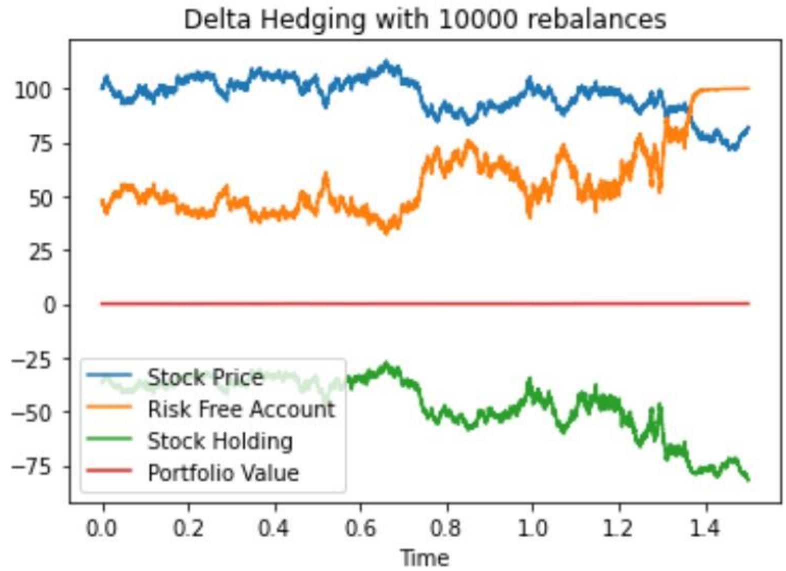
```
K = 100
S0 = 100
mu = 0.08


def deltahedge(N):
    t_list = [0]
    S_list = [100]
    V_list = [blsprice(100,K,r,T,sigma)[1]]
    alpha_list = [blsdelta(100,K,r,T,sigma)[1]]
    B_list = [V_list[0]-alpha_list[0]*S_list[0]]
    holding_list = [alpha_list[0]*S_list[0]]
    Pi_list = [-V_list[0]+alpha_list[0]*S_list[0]+B_list[0]]

    delt_t = T/N
    i = 1
    while i <= N:
        t_list.append(t_list[i-1]+delt_t)
        S_list.append(S_list[i-1]*math.exp((mu-sigma**2/2)*delt_t+sigma*np.
 ↪random.normal(0,1)*math.sqrt(delt_t)))
        V_list.append(blsprice(S_list[i],K,r,T-(i-1)*delt_t,sigma)[1])
        alpha_list.append(blsdelta(S_list[i],K,r,T-(i-1)*delt_t,sigma)[1])
        B_list.append(math.exp(r*delt_t)*B_list[i-1] -␣
 ↪S_list[i]*(alpha_list[i]-alpha_list[i-1]))
        holding_list.append(alpha_list[i]*S_list[i])
        Pi_list.append(-V_list[i]+alpha_list[i]*S_list[i]+B_list[i])
        i = i + 1
    rel_error=math.exp(-r*T)*Pi_list[N]/abs(V_list[0])

    ## plot
    plt.plot(t_list,S_list,label='Stock Price')
    plt.plot(t_list,B_list,label='Risk Free Account')
    plt.plot(t_list,holding_list,label='Stock Holding')
    plt.plot(t_list,Pi_list,label='Portfolio Value')
    plt.title('Delta Hedging with %d rebalances'%(N))
    plt.xlabel('Time')
    plt.legend()
    plt.show()


def get_rel_error(N):
    t_list = [0]
    S_list = [100]
    V_list = [blsprice(100,K,r,T,sigma)[1]]
    alpha_list = [blsdelta(100,K,r,T,sigma)[1]]
    B_list = [V_list[0]-alpha_list[0]*S_list[0]]
    holding_list = [alpha_list[0]*S_list[0]]
```

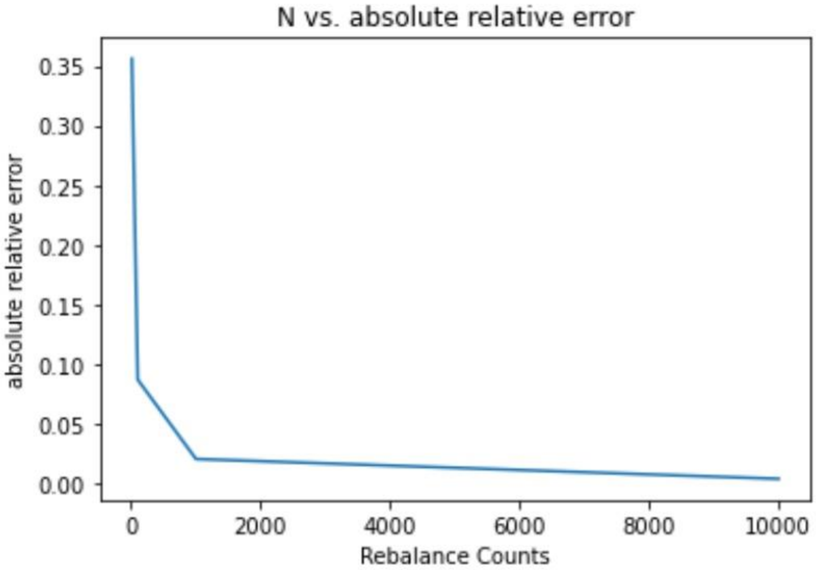
```
    Pi_list = [-V_list[0]+alpha_list[0]*S_list[0]+B_list[0]]
    delt_t = T/N
    i = 1
    while i <= N:
        t_list.append(t_list[i-1]+delt_t)
        S_list.append(S_list[i-1]*math.exp((mu-sigma**2/2)*delt_t+sigma*np.
 ↪random.normal(0,1)*math.sqrt(delt_t)))
        V_list.append(blsprice(S_list[i],K,r,T-(i-1)*delt_t,sigma)[1])
        alpha_list.append(blsdelta(S_list[i],K,r,T-(i-1)*delt_t,sigma)[1])
        B_list.append(math.exp(r*delt_t)*B_list[i-1] -␣
 ↪S_list[i]*(alpha_list[i]-alpha_list[i-1]))
        holding_list.append(alpha_list[i]*S_list[i])
        Pi_list.append(-V_list[i]+alpha_list[i]*S_list[i]+B_list[i])
        i = i + 1
    rel_error=math.exp(-r*T)*Pi_list[N]/abs(V_list[0])
    return rel_error

deltahedge(10)
deltahedge(100)
deltahedge(1000)
deltahedge(10000)

N_list=[10,100,1000,10000]
abs_rel_error_list=[abs(get_rel_error(10)),abs(get_rel_error(100)),
                    abs(get_rel_error(1000)),abs(get_rel_error(10000))]
plt.plot(N_list, abs_rel_error_list)
plt.title('N vs. absolute relative error')
plt.xlabel('Rebalance Counts')
plt.ylabel('absolute relative error')
plt.show()
```


Delta Hedging with 10 rebalances
Stock Price
Risk Free Account
Stock Holding
Portfolio Value
125
100
75
50
25
0
−25
−50
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
Time


Delta Hedging with 100 rebalances
Stock Price
Risk Free Account
Stock Holding
Portfolio Value
125
100
75
50
25
0
−25
−50
−75
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
Time


Delta Hedging with 1000 rebalances
Stock Price
Risk Free Account
Stock Holding
Portfolio Value
Time


Delta Hedging with 10000 rebalances
Stock Price
Risk Free Account
Stock Holding
Portfolio Value
Time



Comments: We can see that the plot of risk free value is symmetric with the stock holding value about the protfolio value (which nearly remains unchanged at 0) in the first four plots. In the plot of N vs abosulte relative error, we can observe that the absolute relative error generally becomes smaller and approch 0 as the reblance counts increase.

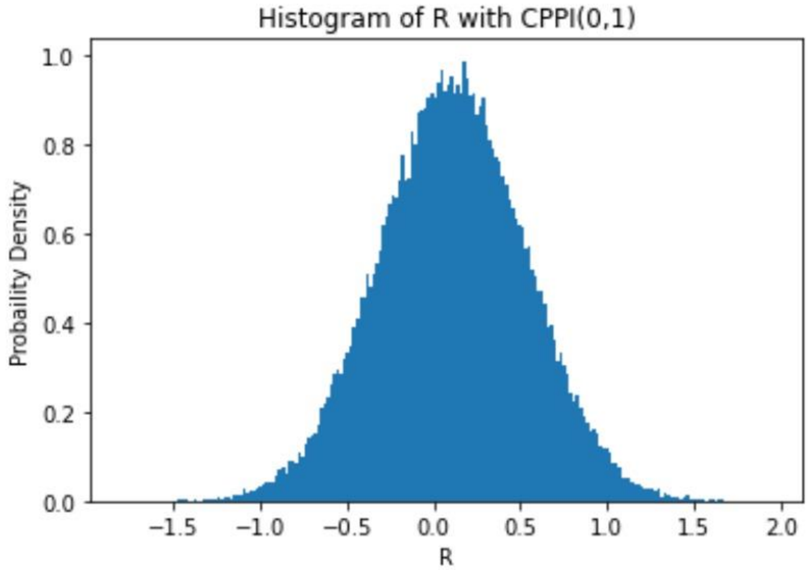
[9]:

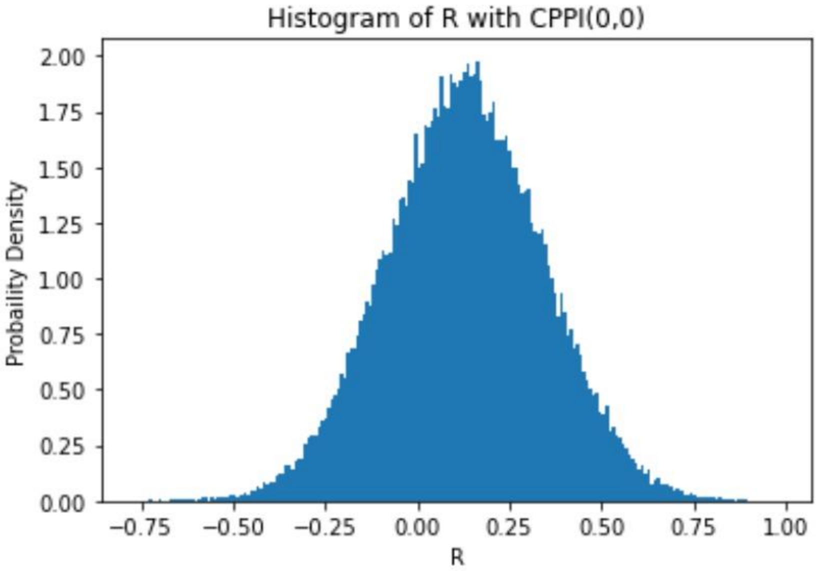
```
## A4Q3
from scipy.stats import norm
import numpy as np
import matplotlib.pyplot as plt
import math
from astropy.table import Table

T=2.0
sigma=0.3
mu=0.1
P0=100
r=0.05
delt_t=1/250
cash_init=100
alpha_start=0
S0=100
num_sim=80000

def hist_CPPI(F,M):
    alpha0=M*max(0,100*np.exp(r*delt_t+alpha_start*S0-F)/S0)
    B0=100*np.exp(r*delt_t)-(alpha0-alpha_start)*S0
    Pi0=100*np.exp(r*delt_t)+alpha_start*S0

    S=S0*np.ones(num_sim)
    alpha=alpha0*np.ones(num_sim)
    B=B0*np.ones(num_sim)
    Pi=Pi0*np.ones(num_sim)
    N=T/delt_t
    i=1
    while i <= N:
        S=S*np.exp((mu-sigma**2/2)*delt_t+(np.random.
↪normal(0,1,num_sim)*sigma*math.sqrt(delt_t)))
        alpha_pre=alpha
        alpha=M*np.maximum(0,B*np.exp(r*delt_t)+alpha*S-F)/S
        B=np.exp(r*delt_t)*B-(alpha-alpha_pre)*S
        Pi=B+alpha_pre*S
        i=i+1
```

```
    R=np.log(Pi/Pi0)
    plt.hist(R,bins=200,density=True)
    plt.xlabel('R')
    plt.ylabel('Probaility Density')
    plt.title('Histogram of R with CPPI(%d,%d)'%(F,M))
    plt.show()


def CPPI(F,M):
    alpha0=M*max(0,100*np.exp(r*delt_t+alpha_start*S0-F)/S0)
    B0=100*np.exp(r*delt_t)-(alpha0-alpha_start)*S0
    Pi0=100*np.exp(r*delt_t)+alpha_start*S0
    S=S0*np.ones(num_sim)
    alpha=alpha0*np.ones(num_sim)
    B=B0*np.ones(num_sim)
    Pi=Pi0*np.ones(num_sim)
    N=T/delt_t
    i=1
    while i <= N:
        S=S*np.exp((mu-sigma**2/2)*delt_t+(np.random.
 ↪normal(0,1,num_sim)*sigma*math.sqrt(delt_t)))
        alpha_pre=alpha
        alpha=M*np.maximum(0,B*np.exp(r*delt_t)+alpha*S-F)/S
        B=np.exp(r*delt_t)*B-(alpha-alpha_pre)*S
        Pi=B+alpha_pre*S
        i=i+1
    R=np.log(Pi/Pi0)
    sortR=np.sort(R)
    meanR=np.mean(R)
    stdR=np.std(R)
    VAR=np.quantile(sortR,0.05)
    CVAR=np.mean(sortR[0:4000])
    return meanR, stdR, VAR, CVAR

F=[0,0,0,85,85]
M=[1,0.5,2,2,4]

j=0
meanR_list=[]
stdR_list=[]
VAR_list=[]
CVAR_list=[]
while j <=4:
    CPPI(F[j],M[j])
    meanR_list.append(round(CPPI(F[j],M[j])[0],6))
    stdR_list.append(round(CPPI(F[j],M[j])[1],6))
    VAR_list.append(round(CPPI(F[j],M[j])[2],6))
```

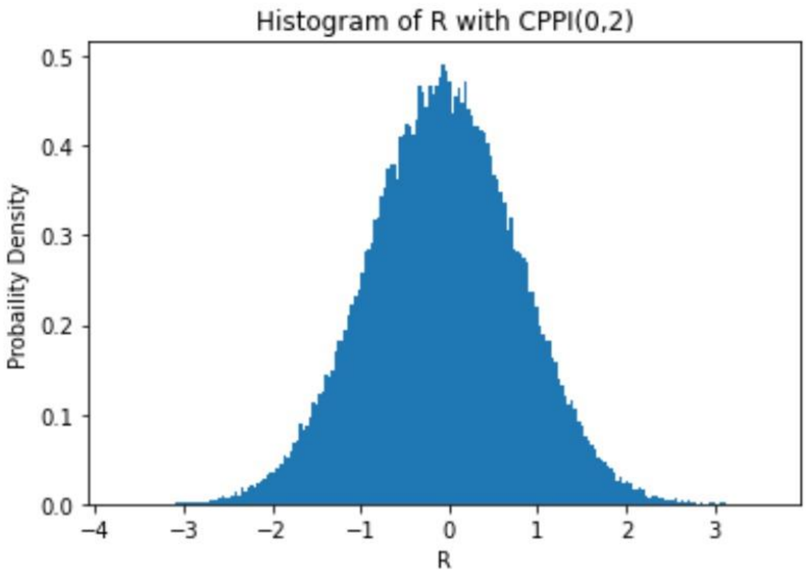
```
    CVAR_list.append(round(CPPI(F[j],M[j])[3],6))
    j=j+1


## draw the table:
Rtable = Table([F,M,meanR_list,stdR_list,VAR_list,CVAR_list],
                      names=('F','M','mean','std','95% VAR','95% CVAR'))
print(Rtable)

hist_CPPI(F[0],M[0])
hist_CPPI(F[1],M[1])
hist_CPPI(F[2],M[2])
hist_CPPI(F[3],M[3])
hist_CPPI(F[4],M[4])
```

```
 F   M     mean      std     95% VAR   95% CVAR
--- --- --------- -------- --------- ---------
  0 1.0  0.112453 0.424626 -0.591293 -0.765044
  0 0.5   0.12576 0.213421 -0.219964 -0.307728
  0 2.0 -0.059231 0.845867  -1.46433 -1.799845
 85 2.0  0.122669 0.180421 -0.072711 -0.089385
 85 4.0   0.09682 0.332923 -0.134993 -0.141869
```

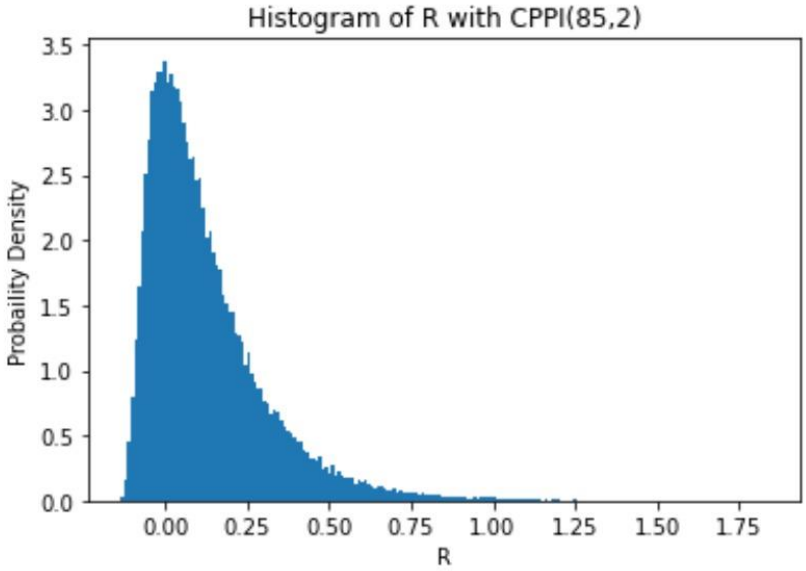


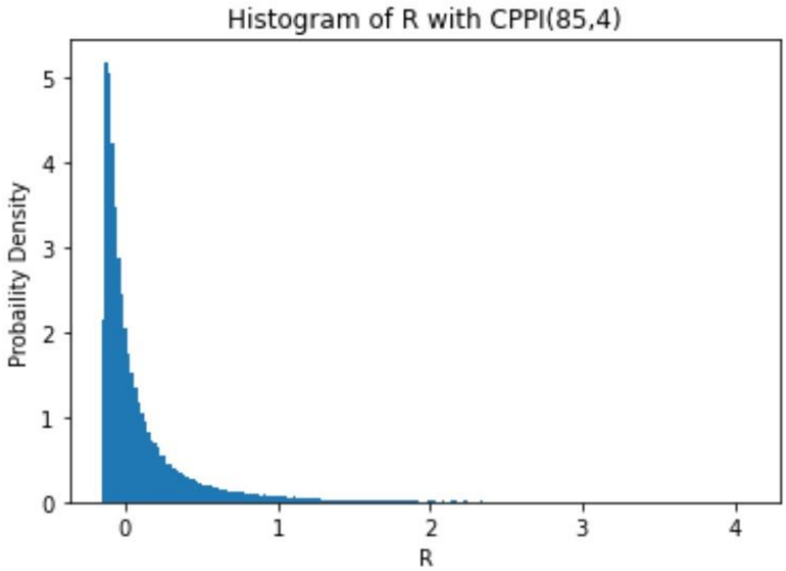


Histogram of R with CPPI(0,0)
Probaility Density
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
R



Histogram of R with CPPI(0,2)
Probaility Density
0.5
0.4
0.3
0.2
0.1
0.0
−4
−3
−2
−1
0
1
2
3
R


Histogram of R with CPPI(85,2)
Probaility Density
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
R

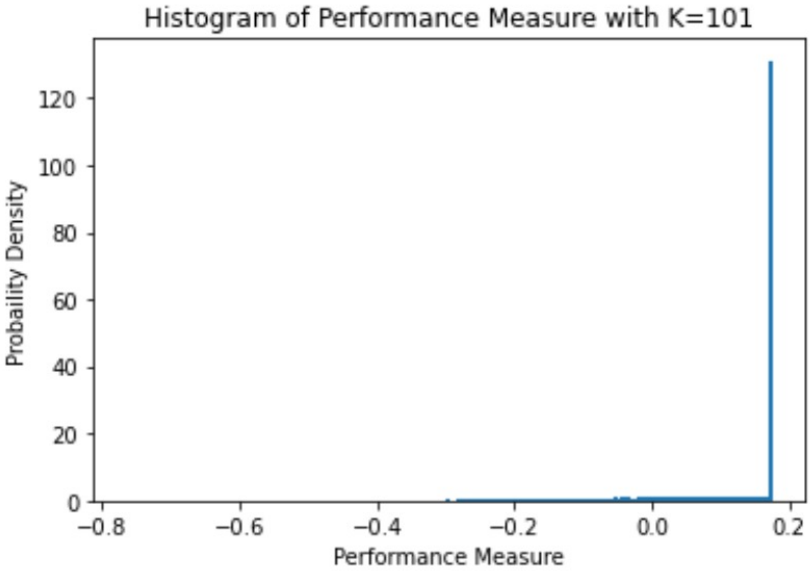

Histogram of R with CPPI(85,4)
Probaility Density
5
4
3
2
1
0
0
1
2
3
4
R

[12]:

```python
## A4Q4
from scipy.stats import norm
import numpy as np
import matplotlib.pyplot as plt
import math
from astropy.table import Table

#adapted from Octave's financial toolkit
def blsprice(Price, Strike, Rate, Time, Volatility):
    sigma_sqrtT = Volatility * np.sqrt (Time)
    d1 = 1 / sigma_sqrtT * (np.log(Price / Strike) + (Rate + Volatility**2 / 2)␣
↪* Time)
    d2 = d1 - sigma_sqrtT
    phi1 = norm.cdf(d1)
    phi2 = norm.cdf(d2)
    disc = np.exp (-Rate * Time)
    F    = Price * np.exp ((Rate) * Time)
    Call = disc * (F * phi1 - Strike * phi2)
    Put  = disc * (Strike * (1 - phi2) + F * (phi1 - 1))
    return Call, Put

sigma=0.25
r=0.05
T=1.75
K_list=[101,115,140]
S0=100
S_init=100
mu=0.09
num_sim=80000

def cover_call(K):
    S_T=np.ones(num_sim)*S0*np.exp((mu-sigma**2/2)*T+(np.random.
↪normal(0,1,num_sim)*sigma*math.sqrt(T)))
    V0=blsprice(S0,K,r,T,sigma)[0]
    payoff=np.maximum(np.zeros(num_sim),S_T-K)
    B0=np.ones(num_sim)*(S_init-S0+V0)
    saving_T=np.exp(r*T)*B0
```

```python
    final_value=saving_T+S_T-payoff
    perf=np.log((final_value/S0))
    ## to get the four outputs:
    sort_perf=np.sort(perf)
    mean_perf=round(np.mean(perf),6)
    std_perf=round(np.std(perf),6)
    VAR=round(np.quantile(sort_perf,0.05),6)
    CVAR=round(np.mean(sort_perf[0:4000]),6)
    return mean_perf,std_perf,VAR,CVAR


def hist_perf(K):
    S_T=np.ones(num_sim)*S0*np.exp((mu-sigma**2/2)*T+(np.random.
 ↪normal(0,1,num_sim)*sigma*math.sqrt(T)))
    V0=blsprice(S0,K,r,T,sigma)[0]
    payoff=np.maximum(np.zeros(num_sim),S_T-K)
    B0=np.ones(num_sim)*(S_init-S0+V0)
    saving_T=np.exp(r*T)*B0
    final_value=saving_T+S_T-payoff
    perf=np.log((final_value/S0))
    ## plot the histogram:
    plt.hist(perf,bins=200,density=True)
    plt.xlabel('Performance Measure')
    plt.ylabel('Probaility Density')
    plt.title('Histogram of Performance Measure with K=%d'%(K))
    plt.show()


mean_list=[cover_call(K_list[0])[0],cover_call(K_list[1])[0],cover_call(K_list[2])[0]]
std_list=[cover_call(K_list[0])[1],cover_call(K_list[1])[1],cover_call(K_list[2])[1]]
VAR_list=[cover_call(K_list[0])[2],cover_call(K_list[1])[2],cover_call(K_list[2])[2]]
CVAR_list=[cover_call(K_list[0])[3],cover_call(K_list[1])[3],cover_call(K_list[2])[3]]


## draw the table:
perf_table = Table([K_list,mean_list,std_list,VAR_list,CVAR_list],
                   names=('K','mean','std','95% VAR','95% CVAR'))
print(perf_table)

## plot the histogram for K = 101:
hist_perf(K_list[0])
```

```
 K    mean      std      95% VAR   95% CVAR
--- -------- -------- --------- ---------
101 0.101453 0.130257 -0.192926 -0.296822
115 0.103764 0.179877 -0.274934 -0.389066
140 0.105997 0.249293 -0.360865 -0.492397
```



Comments: We can observe that the graph is highly left skewed and the measure of performance values are mainly around 0.18 (with apprantly greatest probability density), and the value's mean and sandard deviation tend to increase as K value increases, while the 95% VAR and cVAR tend to decrease at the same time.