In [1]:
```
import numpy as np
from scipy.stats import norm
```

In [2]:
```
a=(np.log(50/50)+.02-0.05)/.2
C=(norm.cdf(.2-a))
```

In [3]:
```
#1.a
def del_eurocall(h):
    sig=0.2
    alpha=0.05
    T=1
    K=50
    So=50
    x=np.random.normal(size=1000000)
    Shu=(So+h)*np.exp((alpha-(sig**2)/2)*T+sig*np.sqrt(T)*x)
    Shd=(So-h)*np.exp((alpha-(sig**2)/2)*T+sig*np.sqrt(T)*x)
    Bhu=np.exp(-alpha*T)*(Shu-K)
    Bhd=np.exp(-alpha*T)*(Shd-K)
    Chu=sum(np.where(Bhu>0,Bhu,0))
    Chd=sum(np.where(Bhd>0,Bhd,0))
    print((Chu-Chd)/(2000000*h))
```

In [4]:
```
del_eurocall(1)
```

```
0.6359768193681798
```

In [5]:
```
del_eurocall(0.1)
```

```
0.6365888758254796
```

In [6]:
```
del_eurocall(0.01)
```

```
0.6365359083916526
```

In [7]:
```
#1.b
def patheurocall():
    So=50
    K=50
    alpha=0.05
    sig=0.2
    T=1
    x=np.random.normal(size=1000000)
    S=(So)*np.exp((alpha-sig**2/2)*T+sig*np.sqrt(T)*x)
    ST=np.exp(-alpha*T)*sum(np.where((S-K)>0,S,0))
    print((ST)/(1000000*So))
```

In [8]:
```
patheurocall()
```

```
0.6364603598536098
```

In [10]:
```
def euroJDP(K):
    sig2=1
    T=4
    a=0
    b2=0.5
    alpha=0.05
    mu=alpha-(np.exp(a+b2/2)-1)
    So=np.repeat(100,100000)
    for j in range(4):
        Z1=np.random.normal(size=100000)
        Z2=np.random.normal(size=100000)
        N=np.random.poisson(size=100000)
        M=a*N+np.sqrt(b2*N)*Z2
        XT=np.log(So)+(mu-sig2/2)+np.sqrt(sig2)*Z1+M
        So=np.exp(XT)

    B=(So-K)
    C=np.exp(-alpha*T)*sum(np.where(B>0,B,0))
    return(C/100000)
```

In [11]:
```
def callJDPvalue(K):

    Call=[]
    for i in range(100):
        Call.append(euroJDP(K))
    np.quantile(Call,q=[0.025,0.975])
    return np.mean(Call)
```

In [12]:
```
callJDPvalue(80)
```

Out[12]:
```
82.92810024195168
```

In [13]:
```
callJDPvalue(100)
```

Out[13]:
```
80.37542105312482
```

In [14]:
```
callJDPvalue(120)
```

Out[14]:
```
78.98064944768939
```

In [16]:
```
def MM1PS(T):
    t=0
    n=0
    mu=1
    NA=0
    ND=0
    A=[]
    D=[]
    N=[]
    TD=np.array([100000000])
    lam=0.8
    ta=-np.log(np.random.uniform(size=1))/lam
    while(ta<=T):
        if(ta<TD.min()):
            t=ta
            NA=NA+1
            n=n+1

            ta=ta-np.log(np.random.uniform(size=1))/lam
            A.append(t)
            if (n>1):
                TD=t+(TD-t)*n/(n-1)
            TD=np.append(TD,t-n*mu*np.log(np.random.uniform(size=1)))
        else:
            t=TD.min()
            n=n-1
            N.append(n)
            ND=ND+1
            D.append(t)

            if (n!=0):
                TD=t+(TD-t)*n/(n+1)
                TD=np.delete(TD,np.argmin(TD))
            else:
                TD=np.array([100000000])
    x=n
    while(n!=0):
        t=TD.min()
        ND=ND+1
        n=n-1
        N.append(n)
        D.append(t)
        if(n>=1):
            TD=np.delete(TD,np.argmin(TD))
            TD=t+(TD-t)*n/(n+1)
    I=np.array(A[1:]).flatten()-np.array(D[:-1])
    Time=np.array(D)-np.array(A).flatten()
    Ti=sum(np.array(D[-x:]))-x*T
    return(sum(np.where(I>0,I,0))/T,(sum(Time)-Ti)/NA,sum(Time)/T)
```

In [18]:
```
MM1PS(100000)
```

Out[18]:
```
(0.19949642509015408, 5.032684421618531, 4.054236698282601)
```

In [17]:
```
Idle=0
average=0
sojourn=0
for i in range(100):
    x,y,z=MM1PS(100000)
    Idle=x+Idle
    sojourn=y+sojourn
    average=z+average
print(Idle/100,average/100,sojourn/100)
```

```
0.2001621658411091 3.9949448164258907 4.994032882260398
```

In [ ]:
```

```