Long_Thanh_NGUYEN_Derivative_Pricing_Assignment3

May 10, 2018

0.1 Assignment 3

May 2018
Long Thanh NGUYEN
long.nguyen2017@qcf.jvn.edu.vn

```
In [1]: !pip install sobol_seq
```

```
Requirement already satisfied: sobol_seq in /Users/longng/anaconda3/lib/python3.6/site-packages
```

1 Question

We consider the problem of pricing an option with payoff $\phi(S_T)$ where the underlying asset S follows the dynamics (under Q):

$$\frac{dS_t}{S_t} = rdt + \sigma dW_t \tag{1}$$

where the risk-free rate and the volatility coef[U+FB01]cient are all deterministic.
For all numerical applications in this assignment we take
x = 100, r = 0.2, σ = 0.1, T = 1.

1.0.1 Question 1

Justify with a short argument why the price of such an option when $S_t = x$ is given by

$$u(t,x) = e^{-r(T-t)}\mathbb{E}$$

Q[ϕ(S_T)| $S_t = x$].(2)
Answer:
We got following parameters: * $e^{-r(T-t)}$: is the discounted factor for the period time t to T * $\mathbb{E}$
Q[ϕ(S_T)| $S_t = x$]: is the expectation as it is the price option function under the dynamics Q.
The expection of payoff function $\phi(S_T)$ with condition at $S_t = x$

1.0.2 Question 2

Suppose we are able to simulate n values of S_T, $\left\{S_T^{(1)}, ..., S_T^{(n)}\right\}$, cite the theorem allowing us to approximate the price of this option by

$$u(t,x) \approx u^{MC}(t,x) := e^{-r(T-t)} \frac{\sum_{i=1}^{n} \phi(S_T^{(i)})}{n} \tag{3}$$

Give an estimation of the expected value and the standard-deviation of $u^{MC}(t,x)$ from those of $\phi(S_T)$. Provide the 95%-con[U+FB01]dence interval of the estimated price.

Answer:

1.0.3 Question 3

We consider the simulation of S_T given $S_t = x$, show (with help of Ito's lemma) that

$$S_T = S_t e^{(r-\sigma^2/2)(T-t)+\sigma(W_T-W_t)}. \tag{4}$$

Using the property of the standard Brownian motion, describe how to simulate S_T from a random normal distribution.

Answer:

We got Ito process as follow:

$$dx = a(x,t)dt + b(x,t)dW_t \tag{5}$$

Consequence we got $G = G(x,t)$:

$$dG = (\frac{\partial G}{\partial x}a + \frac{\partial G}{\partial t} + \frac{1}{2}\frac{\partial^2 G}{\partial x^2}b^2)dt + \frac{\partial G}{\partial x}bdW_t \tag{6}$$

Let $x = S_t, a = rx = rS_t, b = \sigma x = \sigma S_t, G = ln(x) = ln(S_t)$

$$\blacksquare\Rightarrow \begin{cases} \frac{\partial G}{\partial x} = \frac{\partial ln(S_t)}{\partial S_t} = \frac{1}{S_t} \Rightarrow \frac{\partial^2 G}{\partial x^2} = -\frac{1}{S_t^2} \\ \frac{\partial G}{\partial t} = \frac{\partial ln(S_t)}{\partial t} = 0 \end{cases} \tag{7}$$

Then

$$dG = dln(S_t) = (\frac{1}{S_t}rS_t + 0 - \frac{1}{2}\frac{1}{S_t^2}\sigma^2 S_t^2)dt + \frac{1}{S_t}\sigma S_t dW_t \tag{8}$$

$$\Leftrightarrow dln(S_t) = (r - \frac{\sigma^2}{2})dt + \sigma dW_t \tag{9}$$

$$\Rightarrow \int_t^T dln(S_k) = \int_t^T (r - \frac{\sigma^2}{2})dk + \int_t^T \sigma dW_k \tag{10}$$

$$\Leftrightarrow \int_t^T dln(S_k) = (r - \frac{\sigma^2}{2})\int_t^T dk + \sigma\int_t^T dW_k \tag{11}$$

$$\Rightarrow ln\frac{S_T}{S_t} = (r - \frac{\sigma^2}{2})(T-t) + \sigma(W_T - W_t) \tag{12}$$

$$\Leftrightarrow S_T = S_t e^{(r-\sigma^2/2)(T-t)+\sigma(W_T-W_t)}. \tag{13}$$

Simulate S_T as a random normal distribution:

- We got all the input: $r = 0.2, \sigma = 0.1, T = 1, S_t = x = 100, t$
- Generate $(W_T - W_t) \sim N(0, T - t)$
- Run $S_T = S_t e^{(r-\sigma^2/2)(T-t)+\sigma(W_T-W_t)}$

1.0.4 Question 4

In the previous question we directly simulate the normal distribution, show that if we only have the uniform $[0, 1]$ distribution, we would have

$$Z = \mathcal{N}^{-1}(U) \sim N(0,1), U \sim U(0,1), \tag{14}$$

where $\mathcal{N}^{-1}$ is the inverse of the standard normal distribution C.D.F.

Answer:

We got

$$\begin{cases} U \sim U(0,1) \\ Z = \mathcal{N}^{-1}(U) \sim N(0,1) \text{ (standard normal C.D.F)} \end{cases} \tag{15}$$

$$\Rightarrow F_z(z) = P(Z \leq z) = P(\mathcal{N}^{-1}(U) \leq z) = P(U \leq \mathcal{N}(z)) = \mathcal{N}(z) \tag{16}$$

So we can simulate the normal distribution from Uniform Distribution

```
In [2]: import math
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        from scipy import stats
        from scipy.stats import norm
        import sobol_seq as sb
        import pylab as pl
```

1.0.5 Question 5

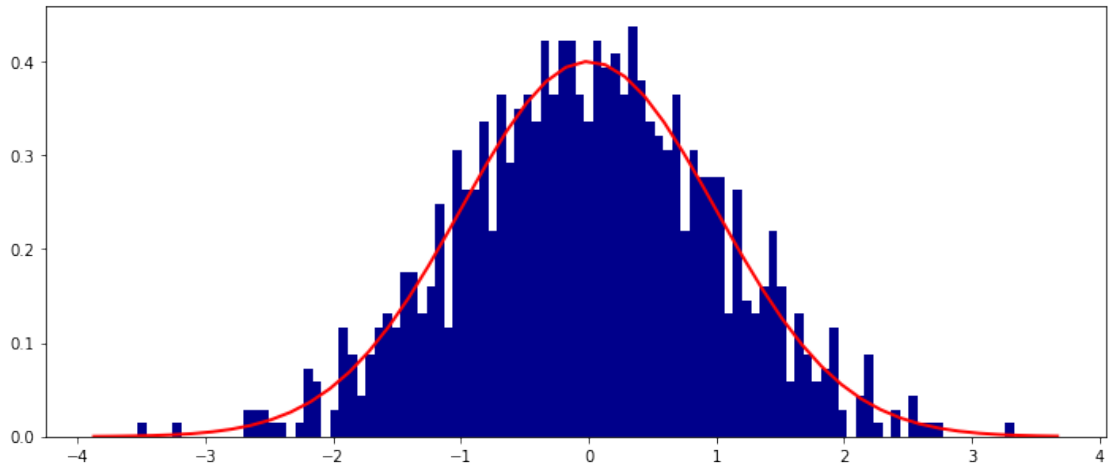
Instead of simulate the true uniform variable, one is able also to use a low discrepancy sequence, for instance, the Sobol sequence - see this link. Apply the previous method to generate n values (n $\in$ {1000,10000,100000}) for the normal distribution, verify indeed that when n becomes large the distribution of generated values converge to the standard normal distribution.

Answer:

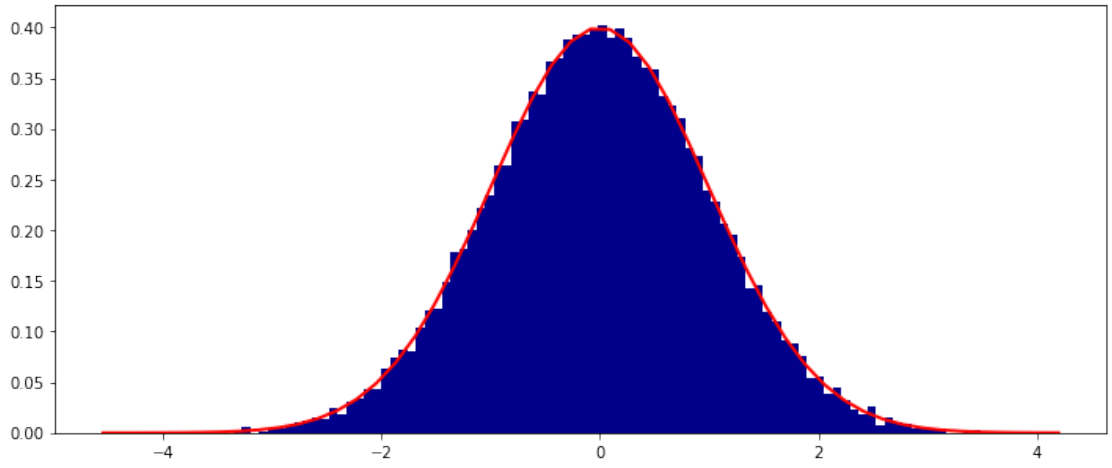
```
In [3]: #generate Uniform distribution by Sobol
        U=sb.i4_sobol_generate(2, 1000)
        UU=sb.i4_sobol_generate(2, 10000)
        UUU=sb.i4_sobol_generate(2, 100000)
```

```
In [4]: #Simulate Normal Distribution from sobol list
        N=np.sqrt(-2*np.log(U[:,0]))* np.cos(2*np.pi*U[:,1])
        NN=np.sqrt(-2*np.log(UU[:,0]))* np.cos(2*np.pi*UU[:,1])
        NNN=np.sqrt(-2*np.log(UUU[:,0]))* np.cos(2*np.pi*UUU[:,1])
```

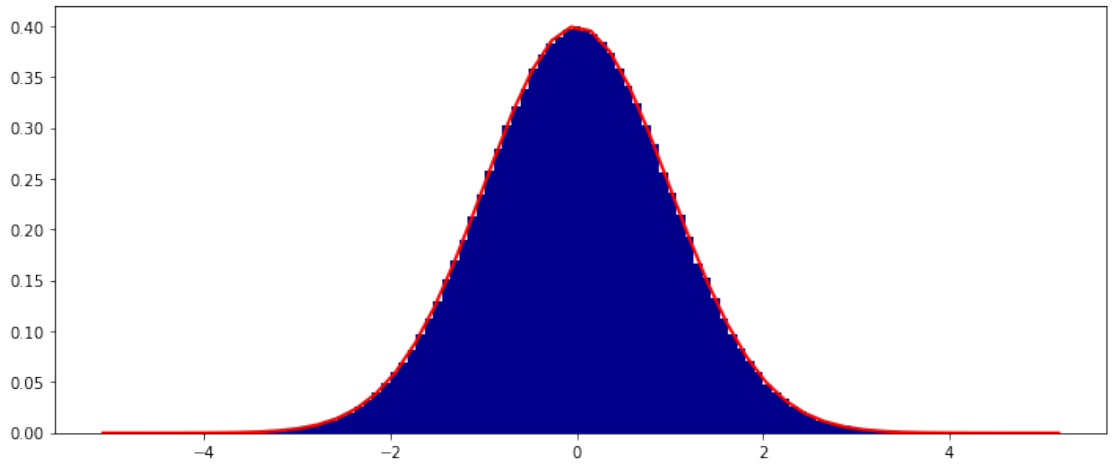
In [5]:
```
#plot the new Normal Distribution
plt.figure(figsize = (12,5))
plt.hist(N,bins=100, normed=True,
color='darkblue')
mu, std = stats.norm.fit(N) //
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax)
p = norm.pdf(x, mu, std)
plt.plot(x, p, 'red', linewidth=2)
plt.show()
```



In [6]:
```
#plot the new Normal Distribution
plt.figure(figsize = (12,5))
plt.hist(NN,bins=100, normed=True, color='darkblue')
mu, std = stats.norm.fit(N)
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax)
p = norm.pdf(x, mu, std)
plt.plot(x, p, 'red', linewidth=2)
plt.show()
```



In [7]:
```
#plot the new Normal Distribution
plt.figure(figsize = (12,5))
plt.hist(NNN,bins=100, normed=True, color='darkblue')
mu, std = stats.norm.fit(N)
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax)
p = norm.pdf(x, mu, std)
plt.plot(x, p, 'red', linewidth=2)
plt.show()
```



Conclusion When n becomes larger and larger, the distribution of generated values converge to the standard normal distribution

1.0.6 Question 6

For the purpose of hedging, we also need (among many greeks) the delta, i.e., ■ = $\frac{\partial u}{\partial x}$. Provide two schemes to approximate ■(t,x) from u(t,x), u(t,x$\pm\varepsilon$)

Answer:

We got Taylor Expansion:

$$f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + ... \tag{17}$$

Let the payoff function as u(t,x) with fixed t, then

$$\blacksquare u(t,x) = \frac{\partial u}{\partial x} \tag{18}$$

Apply $u(t, x+\varepsilon)$ to Taylor Expansion:

$$u(t,x+\varepsilon) = u(t,x) + \frac{\partial u(t,x)}{1!}(x+\varepsilon-x) + \frac{\partial^2 u(t,x)}{2!}(x+\varepsilon-x)^2 + ... = u(t,x) + \frac{\partial u(t,x)}{1!}\varepsilon + \frac{\partial^2 u(t,x)}{2!}\varepsilon^2 + ... \tag{19}$$

$$\Leftrightarrow u(t,x+\varepsilon) \approx u(t,x) + \frac{\partial u(t,x)}{1!} \text{ (as we approximate the rest of the expansion, } \varepsilon \to \infty) \tag{20}$$

Then

$$\blacksquare u(t,x) = u(t,x) - u(t,x+\varepsilon) = u(t,x) - u(t,x) - \frac{\partial u(t,x)}{1!} = -\frac{\partial u}{\partial x} = \frac{u(t,x+\varepsilon) - u(t,x)}{(x+\varepsilon-x)} \tag{21}$$

$$= \frac{u(t,x+\varepsilon) - u(t,x)}{\varepsilon} \tag{22}$$

we got:

$$\blacksquare u(t,x) \approx \frac{u(t,x+\varepsilon) - u(t,x)}{\varepsilon} \tag{23}$$

Similarly, we apply $u(t, x-\varepsilon)$ to Taylor Expansion:

$$u(t,x-\varepsilon) \approx u(t,x) + \frac{\partial u(t,x)}{1!}(x-\varepsilon-x) \approx u(t,x) - \frac{\partial u(t,x)}{1!}(\varepsilon \to \infty) \tag{24}$$

Then, the $u(t,x) - u(t,x-\varepsilon)$

$$\blacksquare u(t,x) = u(t,x) - u(t,x-\varepsilon) = u(t,x) - u(t,x) + \frac{\partial u(t,x)}{1!} = \frac{\partial u}{\partial x} = \frac{u(t,x) - u(t,x-\varepsilon)}{(x-\varepsilon-x)} \tag{25}$$

$$= \frac{u(t,x) - u(t,x-\varepsilon)}{\varepsilon} \tag{26}$$

In short, we got scheme 1 as:

$$\blacksquare u(t,x) \approx \begin{cases} \frac{u(t,x+\varepsilon)-u(t,x)}{\varepsilon} \\ \frac{u(t,x)-u(t,x-\varepsilon)}{\varepsilon} \end{cases} \tag{27}$$

Similarly, the scheme 2 would be:

$$\blacksquare u(t,x) \approx \frac{u(t,x+\varepsilon) - u(t,x-\varepsilon)}{(x+\varepsilon-x+\varepsilon)} = \frac{u(t,x+\varepsilon) - u(t,x-\varepsilon)}{2\varepsilon} \tag{28}$$

1.0.7 Question 7

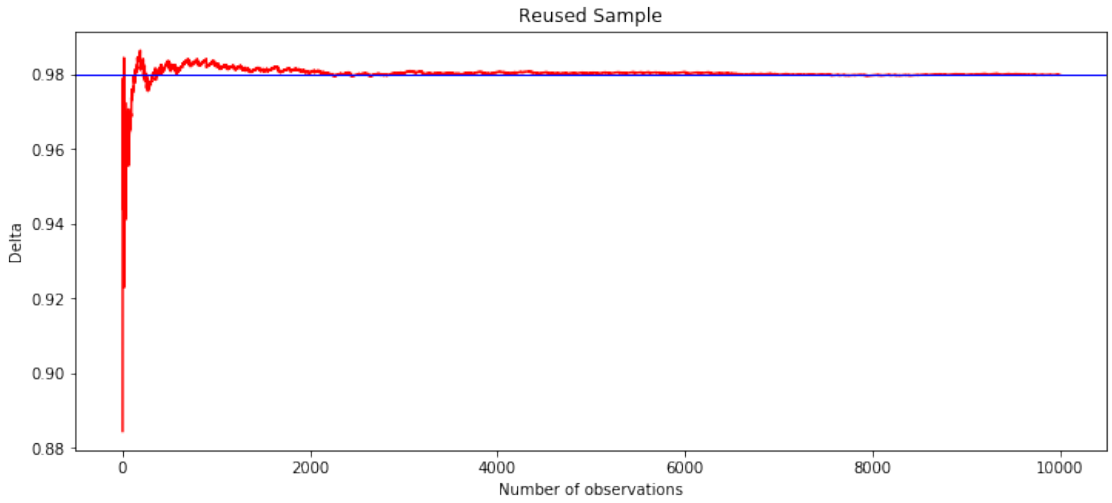
When we approximate $u(t, x \pm \varepsilon)$, we can either simulate new random values or reuse the same random values as when we approximate $u(t, x)$. Verify numerically indeed that it's better to reuse the same random sequences (in comparing for example the rate of convergence to the true value) in the case of an European call option $\phi(S_T) = max(S_T - K, 0)$, For all numerical applications we take $t = 0$, $K = 100$ and $\varepsilon = 0.001$, we also make use of the Sobol sequences.

Answer:

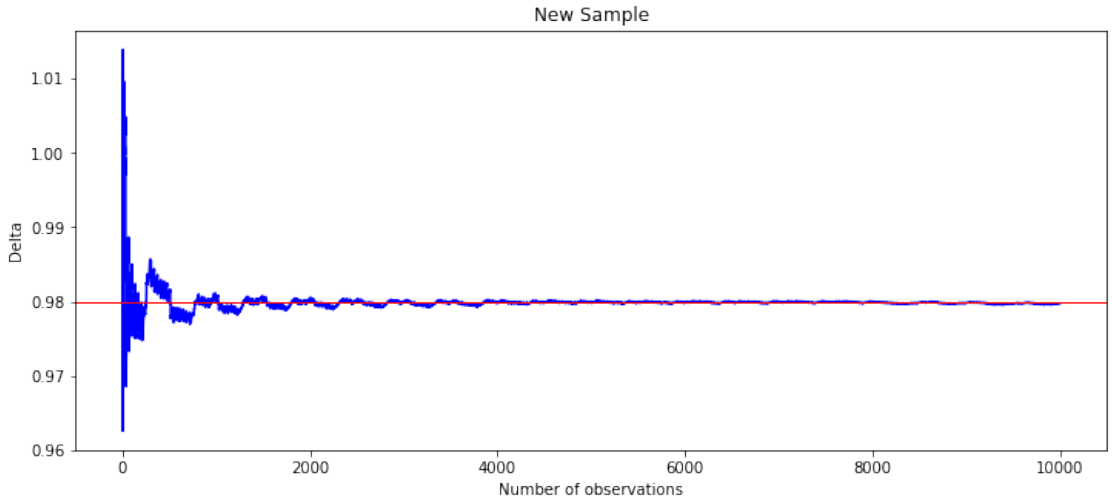
```
In [8]: # Input
        S0 = 100 # Stock Price S0
        K0 = 100 # Strike Price K0
        r0 = 0.2 # Risk free Rate
        sigma0 = 0.1 # Volatility
        T0 = 1 # Time to Maturity
        epsilon0 = 0.001 # Epsilon
        N = 10000

In [9]: #generate the Uniform data sample by sobol and then transform into Normal Distribution
        U1=sb.i4_sobol_generate(2, N)
        U2=sb.i4_sobol_generate(2, N)
        ND1=np.sqrt(-2*np.log(U1[:,0]))* np.cos(2*np.pi*U1[:,1])
        #ND2=np.sqrt(-2*np.log(U2[:,0]))* np.cos(2*np.pi*U2[:,1])
        ND2=stats.norm.ppf(U2[:,1], loc = 0, scale = 1)

    breqn
In [10]: #Option Price Calculation
         StockPrice = lambda t: S0 * np.exp((r0 - 0.5 * (sigma0**2))*T0 + sigma0*np.sqrt(T0)*t)
         StockPricep = lambda t: (S0+epsilon0) * np.exp((r0 - 0.5 * (sigma0**2))*T0 + sigma0*np.
         StockPricem = lambda t: (S0-epsilon0) * np.exp((r0 - 0.5 * (sigma0**2))*T0 + sigma0*np.

         PayOff = lambda t: (abs(t-K0)+t-K0)/2
         OptionPrice = lambda t: np.exp(-r0*T0)*(t)

         s1utx = OptionPrice(PayOff(StockPrice(ND1)))
         s1utxpe = OptionPrice(PayOff(StockPricep(ND1)))
         s1utxme = OptionPrice(PayOff(StockPricem(ND1)))

         s2utxpe = OptionPrice(PayOff(StockPricep(ND2)))
         s2utxme = OptionPrice(PayOff(StockPricem(ND2)))

In [11]: #Delta Caculation for reuse and create new
         #reused
         Dtreuseds1a = (s1utxpe-s1utx)/(epsilon0) # Scheme 1a
         Dtreuseds1a = (s1utx-s1utxme)/(epsilon0) # Scheme 1b
         Dtreuseds2 = (s1utxpe-s1utxme)/(2*epsilon0) # Scheme 2

         #new
         Dtnews1a = (s2utxpe-s1utx)/(epsilon0) # Scheme 1a
```

```
Dtnews1a = (s1utx-s2utxme)/(epsilon0) # Scheme 1b
Dtnews2 = (s2utxpe-s2utxme)/(2*epsilon0) # Scheme 2

BSMP = (np.log(S0/K0) + (r0 + 0.5*(sigma0**2))*T0)/(sigma0*np.sqrt(T0))
TrueDelta = stats.norm.cdf(BSMP)
```

In [12]:
```
#Reused Sample
NX = np.linspace(1, 10000,10000,dtype=np.int)
plt.figure(figsize=(12,5))

plt.title('Reused Sample')
plt.xlabel('Number of observations')
plt.ylabel('Delta')

plt.plot(np.cumsum(Dtreuseds1a)/np.arange(1,10001), color = 'r')
plt.axhline(TrueDelta, color='blue',linewidth=1)
plt.show()
```



In [13]:
```
#New Sample
NX = np.linspace(1, 10000,10000,dtype=np.int)
plt.figure(figsize=(12,5));

plt.title('New Sample')
plt.xlabel('Number of observations')
plt.ylabel('Delta')

plt.plot(np.cumsum(Dtnews2)/np.arange(1,10001), color = 'b')
plt.axhline(TrueDelta, color='red',linewidth=1)
plt.show()
```



1.0.8 Question 8

The FDM scheme to calculate ■ is useful only when the payoff function is smooth (as in European call/put case). When it's not the case, for example a binary option, then FDM method converges very slow to the exact value. Verify that in the case of a binary call option $\phi(S_T) = \mathbb{1}_{S_T \geq K}$ the estimation error is very large even with $n = 10^6$. The estimation error is calculated as

$$\text{err in percentage} = 100 \times \left| \frac{MCvalue - exactvalue}{exactvalue} \right| \tag{29}$$

```
In [14]: # Input
         S0 = 100 # Stock Price S0
         K0 = 100 # Strike Price K0
         r0 = 0.2 # Risk free Rate
         sigma0 = 0.1 # Volatility
         T0 = 1 # Time to Maturity
         epsilon0 = 0.001 # Epsilon
         Nn = 10**6

         #generate data sample
         BiND1 = np.random.normal(0,1,Nn)
         BiND2 = np.random.normal(0,1,Nn)

         BiStockPrice = lambda t: (S0) * np.exp((r0 - 0.5 * (sigma0**2))*T0 + sigma0*np.sqrt(T0)
         BiStockPricep = lambda t: (S0+epsilon0) * np.exp((r0 - 0.5 * (sigma0**2))*T0 + sigma0*n
         BiStockPricem = lambda t: (S0-epsilon0) * np.exp((r0 - 0.5 * (sigma0**2))*T0 + sigma0*n

         BiPayOff = lambda t: np.where(t>K0,1,0)
         BiOptionPrice = lambda t: np.exp(-r0*T0)*np.mean(t)

         Bis1utx = BiOptionPrice(BiPayOff(BiStockPrice(BiND1)))
```

```
        Bis2utxpe = BiOptionPrice(BiPayOff(BiStockPricep(BiND1)))
        Bis2utxme = BiOptionPrice(BiPayOff(BiStockPricem(BiND2)))

        BiDtnews2 = (Bis2utxpe-Bis2utxme)/(2*epsilon0) #reused

        dp = (np.log(S0/K0) + (r0 + 0.5*(sigma0**2))*T0)/(sigma0*np.sqrt(T0))
        #dm = dp - sigma0*np.sqrt(T0)
        BiTrueDelta = stats.norm.cdf(dp)
```

In [15]:
```
# Error Calculation
errip = 100*abs(((BiDtnews2) - BiTrueDelta ) / BiTrueDelta)
print('Error in percentage',errip,'%')
```

```
Error in percentage 96.99185832568254 %
```

1.0.9 Question 9

In such cases, we are able to use an advanced method, the Malliavin Calculus, to calculate the delta (as well as the other greeks). We admit the following result

$$\blacksquare(0, x) = e^{rT}\mathbb{E}^{Q}[\pi$$

$\cdot \phi$(S_T) | $S_0 = x$], π
= W_T$\overline{x\sigma T(30)}$
Compare the estimation error between this case and the previous one.