jgiebas_HW4

February 15, 2018

1 46-932, Simulation Methods for Option Pricing: Homework 4

Author: Jordan Giebas *Due Date*: Feb. 15, 2018

1.1 Question 1: *Practice on Conditional Monte Carlo*

Given the following SDES,

$$dS = rSdt + vSdW_1, \quad dv^2 = \alpha v^2 dt + \psi v^2 dW_2$$

Use an Euler discretization scheme to simulate the paths of the underlying. Do this for two cases, $(\alpha, \psi) = (.10, .10); (0.10, 1.0)$. The parameters are given as,

$S_0 = K = 100$
$r = 0.05$
$T = 1$
$v^2(0) = 0.04$
$N = 50$

1.1.1 Part (a)

Use standard Monte Carlo to estimate the call price and provide standard errors for both cases.

```
In [1]: # Import needed modules
        import pandas as pd
        import numpy as np
        from numpy import exp, sqrt, log
        import scipy
        from scipy.stats import norm
        %matplotlib inline

        # Define problem parameters
        S_0 = K = 100
        r = 0.05
        T = 1.0
        X_0 = 0.04 # Let X = v^2, just keep this in mind as we go
        N = 50.0
        dt = T/N
```

In [2]:
```
def pr1a( alpha, psi ):

    # Construct volatility dataframe
    vol_df = pd.DataFrame(index=pd.RangeIndex(0,51), columns=pd.RangeIndex(1,10001))

    # t0 - v^2 = X = 0.04, populate across columns
    vol_df.loc[0,:] = 0.04

    # Iterate using Euler Scheme
    for i in range(1,len(vol_df.index)):
        Z = np.random.standard_normal(size=10000)
        vol_df.loc[i,:] = vol_df.loc[i-1,:] + alpha*vol_df.loc[i-1,:]*dt + psi*vol_df.l

    # Apply square root to every cell since we need v in underlying evolution
    vol_df = vol_df.applymap(np.sqrt)

    # Construct stock dataframe
    stock_df = pd.DataFrame(index=pd.RangeIndex(0,51), columns=pd.RangeIndex(1,10001))

    # t0 - S_0 = 100
    stock_df.loc[0,:] = S_0

    # Iterate using Euler Scheme
    for i in range(1,len(stock_df.index)):

        Z = np.random.standard_normal(size=10000)
        stock_df.loc[i,:] = stock_df.loc[i-1,:] + r*stock_df.loc[i-1,:]*dt + vol_df.loc

    # Discount the payoffs
    d_payoffs = np.exp(-r*T)*np.maximum(np.array(stock_df.loc[50,:]) - K,0)
    print("Estimated Option Price: $%f \nStandard Error: %f" % (np.mean(d_payoffs),np.s
```

In [3]:
```
# Case I
print("==== CASE I ====")
pr1a(0.10,0.10)

# Case II
print("==== CASE II ====")
pr1a(0.10,1.0)
```

```
==== CASE I ====
Estimated Option Price: $10.518587
Standard Error: 0.150835
==== CASE II ====
Estimated Option Price: $10.504451
Standard Error: 0.156538
```

1.1.2 Part (b)

Use conditional Monte Carlo. I.e., simulate the volatility paths and substitute σ with

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}\sigma^2(i\Delta)}$$

in the Black-Scholes formula to yield the price results.

In [4]:
```
# Define necessary functions for BS (with dividends) price
def d_plus( x, K, tau, sigma, rfr, q ):

    return ( (log(x/K) + ((rfr - q + 0.5*(sigma**2))*tau))/(sigma*sqrt(tau)) )

def d_minus( x, K, tau, sigma, rfr, q ):

    return ( d_plus(x, K, tau, sigma, rfr, q) - sigma*sqrt(tau) )

def BS_price( x, K, r, sigma, tau, q ):

    d_1 = d_plus(x, K, tau, sigma, r, q)
    d_2 = d_minus(x, K, tau, sigma, r, q)

    return ( x*exp(-1.0*q*tau)*norm.cdf(d_1) - K*exp(-1.0*r*tau)*norm.cdf(d_2) )
```

In [5]:
```
def pr1b( alpha, psi ):

    # Construct volatility dataframe
    vol_df = pd.DataFrame(index=pd.RangeIndex(0,52), columns=pd.RangeIndex(1,10001))

    # t0 - v^2 = X = 0.04, populate across columns
    vol_df.loc[0,:] = 0.04

    # Iterate using Euler Scheme
    for i in range(1,len(vol_df.index)-1):
        Z = np.random.standard_normal(size=10000)
        vol_df.loc[i,:] = vol_df.loc[i-1,:] + 0.10*vol_df.loc[i-1,:]*dt + 0.10*vol_df.l

    # Store squared sum as in 51st row
    vol_sq_df = np.square(vol_df).copy()
    vol_sq_df.loc[51,:] = vol_sq_df.loc[1:50,:].sum()/N

    # Get BS prices
    bs_prices = [BS_price(S_0,K,r,np.sqrt(vol_sq_df.loc[51,j]),T,0) for j in range(1,10

    # Report estimates
    print("Conditional MCS Option Price: $%f \nStandard Error: %f" % (np.mean(bs_prices
```

In [6]:
```
# Case I
print("==== CASE I ====")
```

```
pr1b(0.10,0.10)

# Case II
print("==== CASE II ====")
pr1b(0.10,1.0)
```

```
==== CASE I ====
Conditional MCS Option Price: $5.116654
Standard Error: 0.000499
==== CASE II ====
Conditional MCS Option Price: $5.116835
Standard Error: 0.000487
```

There is a variance reduction but we're off in the actual price of the call.

1.1.3 Question 2: *Practice on interest rate derivatives and CIR*

Consider the CIR square root diffusion spot-rate model,

$$dr(t) = \alpha(b - r(t))dt + \sigma\sqrt{r(t)}dW(t)$$

where $t \in [0, T]$. The parameteres are given below:
$\alpha = 0.2$
$\sigma = 0.1$
$b = 0.05$
$r(0) = 0.04$
$N = 50$
$n = 1000$
Where n, N are the number of paths and time steps respectively.

1.1.4 Part (a)

Find the price of a zero coupon bond paying \$1 at time $T = 1$.

In [7]:
```
## Define pr2 parameters
alpha = 0.2
sigma = 0.1
b = 0.05
r_0 = 0.04
n = 1000
N = 50
T = 1
dt = T/N

# Constant model parameter
nu = (4*b*alpha)/(sigma**2)

# Construct the interest rate process dataframe
```

```
        r_df = pd.DataFrame(index=pd.RangeIndex(0,52), columns=pd.RangeIndex(1,1001))
        r_df.iloc[0,:] = r_0

        # Since d > 1, we use Case I referencing Glasserman pg. 124
        for i in range(1,len(r_df)-1):

            ## Dynamic model parameters, since t-s = dt iterating over each time step
            const = ((1-np.exp(-alpha*dt))*(sigma**2))/(4*alpha)
            lam = r_df.loc[i-1,:]*(np.exp(-alpha*dt)/const)

            # Generate Standard Normal, non-central Chi-squared
            #Z = np.random.standard_normal(size=1000)
            X = np.random.noncentral_chisquare(df=nu,nonc=lam.tolist(),size=1000)
            #X = np.random.chisquare(df=nu,size=1000)

            #r_df.loc[i,:] = const*( (Z + np.sqrt(lam.tolist()))**2 + X )
            r_df.loc[i,:] = const*X

        # Aggregate data
        r_df.loc[51,:] = r_df.loc[0:50,:].sum()
        arg = -1.0*dt*np.array(r_df.loc[51,:])
        c_i = np.exp(arg.tolist())

        # Report estimate / stderror
        print("The bond price estimate is %f with a standard error of %f" % (np.mean(c_i),np.st
```

```
The bond price estimate is 0.959369 with a standard error of 0.000335
```

1.1.5 Part (b)

Price the caplet with payoff given below,

$$payoff = L\delta(r(t) - R)^+$$

If the parameters are given by,
$t = 1$
$\delta = \frac{1}{12}$
$L = 1$
$R = 0.05$
and the payoff occurs at $t = 1$.

In [8]:
```
        # Params
        L = 1
        R = 0.05
        delta = 1/12.0

        # Get payoffs
        payoffs = np.array(np.maximum(r_df.loc[50,:] - R,0))
```

```
        # Discount the payoffs
        d_payoffs = np.multiply(np.exp(-1.0*np.array(r_df.loc[49,:].tolist())), payoffs)

        # Report estimate / standard error
        print("The capelt price estimate is %f with a standard error of %f" % (np.mean(d_payof
```

```
The capelt price estimate is 0.000357 with a standard error of 0.000292
```

1.2 Question 3: *Replicating Glasserman's 'Greeks' methodology*

Replicate the Delta and Vega values found in Table 1 on pg. 276 of the Glasserman and Broadie text: report the value and associated standard errors. Perform this replication using the resimulation, pathwise, and likelihood ratio methods. Use the following parameters:

$r = 0.10$
$K = 100$
$\delta = 0.03$
$\sigma = 0.25$
$T = 0.20$
$n = 10000$
$h = 0.0001$

Use the formulas in the Glassman text to simulate the Greeks, and S_T as a control variable during the simulations.

In [9]:
```
# Define the global parameters
r = 0.10
K = 100
q = 0.03
sig = 0.25
T = 0.20
n = 10000
h = 0.0001
S_0 = 90
```

The following four cells use numerical differentiation for the Δ and *Vega* with and without using S_T as a control variable.

In [10]:
```
# Delta no control

        # Generate price process starting from S_0 + h, S_0 - h, central differences
        ppmh_df = pd.DataFrame(index=pd.RangeIndex(start=0,stop=2001),columns=pd.RangeIndex(st
        ppmh_df.loc[0,:]  = S_0 - h
        ppph_df = pd.DataFrame(index=pd.RangeIndex(start=0,stop=2001),columns=pd.RangeIndex(st
        ppph_df.loc[0,:]  = S_0 + h

        # Populate DataFrame
        for i in range(1,len(ppmh_df)-1):
            Z = np.random.standard_normal(size=10000)
```

```
            ppmh_df.loc[i,:] = ppmh_df.loc[i-1,:]*np.exp( (r-q-0.5*(sig**2))*(h) + sig*np.sqr
            ppph_df.loc[i,:] = ppph_df.loc[i-1,:]*np.exp( (r-q-0.5*(sig**2))*(h) + sig*np.sqr

        # Use central differences to get the derivative
        ppmh_df.loc[2000,:]  = np.maximum(ppmh_df.loc[1999,:] - K, 0)
        ppph_df.loc[2000,:] = np.maximum(ppph_df.loc[1999,:] - K, 0)
        central_diff = (np.subtract(np.array(ppph_df.loc[2000,:]), np.array(ppmh_df.loc[2000,

        # Aggregate data
        resim_delta_noncontrol_estimate = np.exp(-r*T)*np.mean(central_diff)
        resim_delta_noncontrol_stderror = np.std(central_diff)/np.sqrt(10000)

        print("Resimulation No-Control Delta: ", resim_delta_noncontrol_estimate)
        print("Resimulation No-Control Delta StdError:", resim_delta_noncontrol_stderror)
```

```
Resimulation No-Control Delta:  0.218005920092
Resimulation No-Control Delta StdError: 0.00462690121465
```

In [11]:
```
        # Delta with Control Variable S_T

        ##  Define a list of all the price processes (list of lists)
        Y_  = central_diff.tolist()
        X_  = ppph_df.loc[1999,:].tolist()

        # Get quantities needed for adjustment
        Y_bar = np.mean(Y_)
        X_bar = np.mean(X_)
        a_hat = -1.0*np.corrcoef(X_,Y_)[0][1]*(np.std(Y_)/np.std(X_))

        # Adjust Y
        Y_adj = Y_bar + a_hat*(S_0*np.exp(r*T) - X_bar)

        # Get estimates
        resim_delta_control_estimate = Y_adj
        resim_delta_control_stderror = (np.std(Y_)/np.sqrt(10000))*np.sqrt(1-(np.corrcoef(X_,
```

```
        # Report results
        print( "Resimulation Control Delta: ", resim_delta_control_estimate )
        print( "Resimulation Control Delta Std. Error: ", resim_delta_control_stderror )
```

```
Resimulation Control Delta:  0.204506227855
Resimulation Control Delta Std. Error:  0.00314914854215
```

In [12]:
```
        # Vega without Control

        # Generate price process starting from S_0 + h, S_0 - h, central differences
        ppmh_vol_df = pd.DataFrame(index=pd.RangeIndex(start=0,stop=2001),columns=pd.RangeInde
```

```
        ppmh_vol_df.loc[0,:]  = S_0
        ppph_vol_df = pd.DataFrame(index=pd.RangeIndex(start=0,stop=2001),columns=pd.RangeInde
        ppph_vol_df.loc[0,:]  = S_0

        # Populate the dataframes
        for i in range(1,len(ppmh_vol_df)-1):
            Z = np.random.standard_normal(size=10000)
            ppmh_vol_df.loc[i,:] = ppmh_vol_df.loc[i-1,:]*np.exp( (r-q-0.5*((sig-h)**2))*(h)
            ppph_vol_df.loc[i,:]  = ppph_vol_df.loc[i-1,:]*np.exp( (r-q-0.5*((sig+h)**2))*(h)

        # Report metrics
        ppmh_vol_df.loc[2000,:]  = np.maximum(ppmh_vol_df.loc[1999,:] - K, 0)
        ppph_vol_df.loc[2000,:] = np.maximum(ppph_vol_df.loc[1999,:] - K, 0)
        central_diff = (np.subtract(np.array(ppph_vol_df.loc[2000,:]), np.array(ppmh_vol_df.lo

        resim_vega_noncontrol_estimate = np.exp(-r*T)*np.mean(central_diff)
        resim_vega_noncontrol_stderror = np.std(central_diff)/np.sqrt(10000)

        print("Resimulation No-Control Vega: ", resim_vega_noncontrol_estimate)
        print("Resimulation No-Control Vega StdError:", resim_vega_noncontrol_stderror)
```

```
Resimulation No-Control Vega:  11.7751582661
Resimulation No-Control Vega StdError: 0.277383902258
```

In [13]:
```
        # Vega with Control Variable S_T

        ##  Define a list of all the price processes (list of lists)
        Y_  = central_diff.tolist()
        X_  = ppph_vol_df.loc[1999,:].tolist()

        # Get quantities needed for adjustment
        Y_bar = np.mean(Y_)
        X_bar = np.mean(X_)

        a_hat = -1.0*np.corrcoef(X_,Y_)[0][1]*(np.std(Y_)/np.std(X_))

        Y_adj = Y_bar + a_hat*(S_0*np.exp(r*T) - X_bar)

        resim_vega_control_estimate = Y_adj
        resim_vega_control_stderror = (np.std(Y_)/np.sqrt(10000))*np.sqrt(1-(np.corrcoef(X_,Y_

        print( "Resimulation Control Vega: ", resim_vega_control_estimate )
        print( "Resimulation Control Vega Std. Error: ", resim_vega_control_stderror )
```

```
Resimulation Control Vega:  11.0833311862
Resimulation Control Vega Std. Error:  0.180902353922
```

The following four cells use pathwise estimates for the Δ and *Vega* with and without using S_T as a control variable.

In [14]:
```
## Pathwise Estimates no control

## Pathwise Delta - no control

# Simulate a price process starting at S_0
pprocess = pd.DataFrame(index=pd.RangeIndex(start=0,stop=2001),columns=pd.RangeIndex(s
pprocess.loc[0,:] = S_0

# Populate the dataframes
for i in range(1,len(pprocess)-1):
    Z = np.random.standard_normal(size=10000)
    pprocess.loc[i,:] = pprocess.loc[i-1,:]*np.exp( (r-q-0.5*(sig**2))*(h) + sig*np.sc

# Aggregate results
pprocess.loc[2000,:] = np.where( pprocess.loc[1999,:] >= K, 1, 0)
S_TbyS_0 = (np.exp(-r*T)/S_0)*pprocess.loc[1999,:]
indicator = pprocess.loc[2000,:]
res = np.multiply(S_TbyS_0,indicator)

# Report
pathwise_delta_noncontrol_estimate = np.exp(-r*T)*np.mean(res)
pathwise_delta_noncontrol_stderror = np.std(res)/np.sqrt(10000)
print("Pathwise No-Control Delta: ", pathwise_delta_noncontrol_estimate)
print("Pathwise No-Control Delta Std. Error: ", pathwise_delta_noncontrol_stderror)
```

```
Pathwise No-Control Delta:  0.221921894961
Pathwise No-Control Delta Std. Error:  0.00460216326986
```

In [15]:
```
## Pathwise Delta Control Variable S_T

##  Define a list of all the price processes (list of lists)
Y_  = res.tolist()
X_  = pprocess.loc[1999,:].tolist()

# Get quantities needed for adjustment
Y_bar = np.mean(Y_)
X_bar = np.mean(X_)
a_hat = -1.0*np.corrcoef(X_,Y_)[0][1]*(np.std(Y_)/np.std(X_))

# Adjust the Y
Y_adj = Y_bar + a_hat*(S_0*np.exp(r*T) - X_bar)

# Report results
resim_vega_control_estimate = Y_adj
```

```
resim_vega_control_stderror = (np.std(Y_)/np.sqrt(10000))*np.sqrt(1-(np.corrcoef(X_,Y
print( "Pathwise Control Delta: ", resim_vega_control_estimate )
print( "Pathwise Control Delta Std.Error: ", resim_vega_control_stderror )
```

```
Pathwise Control Delta:  0.209008147921
Pathwise Control Delta Std.Error:  0.00309942914723
```

In [16]:
```
## Pathwise Vega no control

# Simulate a price process starting at S_0
pw_vol_process = pd.DataFrame(index=pd.RangeIndex(start=0,stop=2001),columns=pd.Range
pw_vol_process.loc[0,:] = S_0

# Populate the dataframes
for i in range(1,len(pw_vol_process)-1):
    Z = np.random.standard_normal(size=10000)
    pw_vol_process.loc[i,:] = pw_vol_process.loc[i-1,:]*np.exp( (r-q-0.5*(sig**2))*(h

# Store indicator row
pw_vol_process.loc[2000,:] = np.where( pw_vol_process.loc[1999,:] >= K, 1, 0)

# Get the payoffs
indicator = pw_vol_process.loc[2000,:]
S_Tbysig = pw_vol_process.loc[1999,:]/sig
term1 = np.log(np.divide(pw_vol_process.loc[1999,:],pw_vol_process.loc[0,:]).tolist()
res = np.exp(-r*T)*np.multiply(np.multiply(indicator, S_Tbysig), term1)

# Report results
pathwise_vega_noncontrol_estimate = np.mean(res)
pathwise_vega_noncontrol_stderror = np.std(res)/np.sqrt(10000)
print("Pathwise No-Control Vega: ", pathwise_vega_noncontrol_estimate)
print("Pathwise No-Control Vega StdError:", pathwise_vega_noncontrol_stderror)
```

```
Pathwise No-Control Vega:  11.7029716835
Pathwise No-Control Vega StdError: 0.268779134119
```

In [17]:
```
## Pathwise Vega control variable S_T

##  Define a list of all the price processes (list of lists)
Y_  = res.tolist()
X_  = pw_vol_process.loc[1999,:].tolist()

# Get quantities needed for adjustment
Y_bar = np.mean(Y_)
X_bar = np.mean(X_)
a_hat = -1.0*np.corrcoef(X_,Y_)[0][1]*(np.std(Y_)/np.std(X_))
```

```
        # Adjust the Y
        Y_adj = Y_bar + a_hat*(S_0*np.exp(r*T) - X_bar)

        # Report results
        pathwise_vega_control_estimate = Y_adj
        pathwise_vega_control_stderror = (np.std(Y_)/np.sqrt(10000))*np.sqrt(1-(np.corrcoef(X_
        print( "Pathwise Control Vega: ", pathwise_vega_control_estimate )
        print( "Pathwise Control Vega Std.Error: ", pathwise_vega_control_stderror )
```

```
Pathwise Control Vega:  10.4050829164
Pathwise Control Vega Std.Error:  0.175758304991
```

The following four cells use LLH for the Δ and *Vega* with and without using S_T as a control variable.

```
In [18]: ## Log Likelihood Method No Control

         ## LLH Delta No Control

         # Simulate a price process starting at S_0
         pp_ = pd.DataFrame(index=pd.RangeIndex(start=0,stop=2001),columns=pd.RangeIndex(start=
         pp_.loc[0,:] = S_0

         # Populate the dataframes
         for i in range(1,len(pprocess)-1):
             Z = np.random.standard_normal(size=10000)
             pp_.loc[i,:] = pp_.loc[i-1,:]*np.exp( (r-q-0.5*(sig**2))*(h) + sig*np.sqrt(h)*Z )

         # Put payoff as last row
         pp_.loc[2000,:] = np.maximum(pp_.loc[1999,:] - K, 0)

         # Get the individual quantities to average over
         term  = np.log(np.divide(pp_.loc[1999,:],pp_.loc[0,:]).tolist()) - (r-q+0.5*sig**2)*T
         term /= S_0*(sig**2)*T
         res = np.exp(-r*T)*np.multiply(np.array(pp_.loc[2000,:]), term)

         # Report results
         llh_delta_noncontrol_estimate = np.mean(res)
         llh_delta_noncontrol_stderror = np.std(res)/np.sqrt(10000)
         print( "LLH non-control Delta: ", llh_delta_noncontrol_estimate )
         print( "LLH non-control Delta Std.Error: ", llh_delta_noncontrol_stderror )
```

```
LLH non-control Delta:  0.2149283594125386
LLH non-control Delta Std.Error:  0.00762051884104
```

```
In [19]: ## LLH Delta Control Variable S_T
```

```
        ##  Define a list of all the price processes (list of lists)
        Y_  = res.tolist()
        X_  = pp_.loc[1999,:].tolist()

        # Get quantities needed for adjustment
        Y_bar = np.mean(Y_)
        X_bar = np.mean(X_)

        a_hat = -1.0*np.corrcoef(X_,Y_)[0][1]*(np.std(Y_)/np.std(X_))
        Y_adj = Y_bar + a_hat*(S_0*np.exp(r*T) - X_bar)

        llh_delta_control_estimate = Y_adj
        llh_delta_control_stderror = (np.std(Y_)/np.sqrt(10000))*np.sqrt(1-(np.corrcoef(X_,Y_

        print( "LLH Control Delta: ", llh_delta_control_estimate )
        print( "LLH Control Delta Std.Error: ", llh_delta_control_stderror )
```

```
LLH Control Delta:  0.199933400249
LLH Control Delta Std.Error:  0.00607078011067
```

In [20]:
```
## LLH Vega No Control

        # Simulate a price process starting at S_0
        pp_ = pd.DataFrame(index=pd.RangeIndex(start=0,stop=2001),columns=pd.RangeIndex(start=
        pp_.loc[0,:] = S_0

        # Populate the dataframes
        for i in range(1,len(pprocess)-1):
            Z = np.random.standard_normal(size=10000)
            pp_.loc[i,:] = pp_.loc[i-1,:]*np.exp( (r-q-0.5*(sig**2))*(h) + sig*np.sqrt(h)*Z )

        # Put payoff as last row
        pp_.loc[2000,:] = np.maximum(pp_.loc[1999,:] - K, 0)

        # Get the individual quantities to average over
        u  = np.log(np.divide(pp_.loc[1999,:],pp_.loc[0,:]).tolist()) - (r-q-0.5*sig**2)*T
        term = (-u/(sig*np.sqrt(T)))*(np.sqrt(T) - u/(sig**2 * np.sqrt(T))) - 1/sig

        res = np.multiply(term, pp_.loc[2000,:])

        llh_vega_noncontrol_estimate = np.exp(-r*T)*np.mean(res)
        llh_vega_noncontrol_stderror = np.std(res)/np.sqrt(10000)

        print( "LLH non-control Vega: ", llh_vega_noncontrol_estimate )
        print( "LLH non-control Vega Std.Error: ", llh_vega_noncontrol_stderror )
```

```
LLH non-control Vega:  11.5923073054
```

```
LLH non-control Vega Std.Error:  0.655069536855
```

```
In [21]: ## LLH Vega Control Variable S_T

         ##  Define a list of all the price processes (list of lists)
         Y_  = res.tolist()
         X_  = pp_.loc[1999,:].tolist()

         # Get quantities needed for adjustment
         Y_bar = np.mean(Y_)
         X_bar = np.mean(X_)

         a_hat = -1.0*np.corrcoef(X_,Y_)[0][1]*(np.std(Y_)/np.std(X_))
         Y_adj = Y_bar + a_hat*(S_0*np.exp(r*T) - X_bar)

         llh_vega_control_estimate = Y_adj
         llh_vega_control_stderror = (np.std(Y_)/np.sqrt(10000))*np.sqrt(1-(np.corrcoef(X_,Y_)

         print( "LLH Control Vega: ", llh_vega_control_estimate )
         print( "LLH Control Vega Std.Error: ", llh_vega_control_stderror )
```

```
LLH Control Vega:  10.5125077788
LLH Control Vega Std.Error:  0.582024528427
```

1.3 Question 4: *Applying Broadie and Glasserman to Digital Options*

Consider a digital option paying \$1 if $S_T \geq K$. Assume the asset price is a GBM under the risk neutral measure. The parameters for this problem are given by the following,

$S_0 = 95$
$K = 100$
$r = 0.05$
$\sigma = 0.20$
$T = 1$
$n = 10000$
$h = 0.0001$

1.3.1 Part (a)

Using the closed-form expression for the price of a digital option, compute the price and delta of this option. **Note:** The delta is given by

$$\frac{dP}{dS} = \frac{d}{dS}\left[e^{-rT}\Phi(-d)\right]$$

where

$$d := \frac{ln\left(\frac{K}{S_0}\right) - (r - \frac{1}{2}\sigma^2)T)}{\sigma\sqrt{T}}$$

Hence,

$$\Delta^{Digital} = \frac{1}{S_0\sigma\sqrt{2\pi T}} exp\{-(\frac{1}{2}d^2 + rT)\}$$

```
In [22]: # Price of digital option
         def digital_option(S_0, T, sigma, r, K):

             d = (np.log(K/S_0) - (r-0.5*sigma**2)*T)/(sigma*np.sqrt(T))

             return np.exp(-r*T)*scipy.stats.norm.cdf(-1.0*d)

         # Delta of digital option
         def delta_digital_option(S_0, T, sigma, r, K):

             d = (np.log(K/S_0) - (r-0.5*sigma**2)*T)/(sigma*np.sqrt(T))

             return np.exp(-0.5*d**2 - r*T)/(S_0*sigma*np.sqrt(2*np.pi*T))
```

```
In [23]: # Define parameters, compute price and delta
         S_0 = 95
         K = 100
         r = 0.05
         sig = 0.20
         T = 1

         print("The price of the option is: ", digital_option(S_0,T,sig,r,K))
         print("The delta of the option is: ", delta_digital_option(S_0,T,sig,r,K))
```

```
The price of the option is:  0.435288413642
The delta of the option is:  0.019860050706
```

1.3.2 Part (b)

Use the resimulation method to estimate the delta of the option. Compare with the closed form expression given part (a).

```
In [24]: h = 0.0001
         ppph_df = pd.DataFrame(index=pd.RangeIndex(start=0,stop=2001),columns=pd.RangeIndex(st
         ppph_df.loc[0,:] = S_0 + h
         ppmh_df = pd.DataFrame(index=pd.RangeIndex(start=0,stop=2001),columns=pd.RangeIndex(st
         ppmh_df.loc[0,:] = S_0 - h

         for i in range(1, len(ppph_df)-1):

             Z = np.random.standard_normal(size=10000)
             ppph_df.loc[i,:] = ppph_df.loc[i-1,:]*np.exp( (r-0.5*(sig**2))*h + sig*np.sqrt(h)*
             ppmh_df.loc[i,:] = ppmh_df.loc[i-1,:]*np.exp( (r-0.5*(sig**2))*h + sig*np.sqrt(h)*
```

```
ppph_df.loc[2000,:] = np.maximum(ppph_df.loc[1999,:]-K,0)
ppmh_df.loc[2000,:] = np.maximum(ppmh_df.loc[1999,:]-K,0)

diff = np.subtract(ppph_df.loc[2000,:], ppmh_df.loc[2000,:])/(2*h)

pr4_b_estimate = np.mean(diff)
pr4_b_stderror = np.std(diff)/np.sqrt(10000)

print("The delta estimate is %f and the standard error is %f" % (np.exp(-r*T)*pr4_b_es
```

```
The delta estimate is 0.324406 and the standard error is 0.005149
```

1.3.3 Part (c)

Use the liklihood method to estimate the delta of the option. Compoare with the closed form expression given in part (a).

```
In [25]: h = 0.0001
         # Simulate a price process starting at S_0
         pp_ = pd.DataFrame(index=pd.RangeIndex(start=0,stop=2001),columns=pd.RangeIndex(start=
         pp_.loc[0,:] = S_0

         # Populate the dataframes
         for i in range(1,len(pp_)-1):
             Z = np.random.standard_normal(size=10000)
             pp_.loc[i,:] = pp_.loc[i-1,:]*np.exp( (r-0.5*(sig**2))*(h) + sig*np.sqrt(h)*Z )

         # Put payoff as last row (digital)
         pp_.loc[2000,:] = np.where(pp_.loc[1999,:] >= K, 1, 0)

         # Get the individual quantities to average over
         term  = np.log(np.divide(pp_.loc[1999,:],pp_.loc[0,:]).tolist()) - (r+0.5*sig**2)*T
         term /= S_0*(sig**2)*T
         res = np.multiply(np.array(pp_.loc[2000,:]), term)

         # Get estimates
         pr4c_estimate = np.mean(res)
         pr4c_stderror = np.std(res)/np.sqrt(10000)

         # Report
         print("The LLH delta estimate is %f and the standard error is %f" % (np.exp(-r*T)*pr4c
```

```
The LLH delta estimate is 0.003038 and the standard error is 0.000085
```