contvar_py

March 20, 2025

0.1 Control Variate Sampling

[1]:
```
import numpy as np
```

[2]:
```
def f(x):
    value = 1 /(1+x)
    return value
```

[3]:
```
def g(x):
    value = 1 + x
    return value
```

[4]:
```
truth = 3.0 / 2.0
```

[5]:
```
truth
```

[5]:
```
1.5
```

0.1.1 The Naive Solution

[21]:
```
n = 15_000
u = np.random.uniform(size=n)
x1 = f(u)
x1.mean()
x1.var()
x1.std()
se = x1.std() / np.sqrt(n)
```

[31]:
```
np.round(np.mean(x1), 4)
```

[31]:
```
np.float64(0.6934)
```

[23]:
```
se
```

[23]:
```
np.float64(0.0011406080903478744)
```

0.1.2 The Control Variate Solution

[24]:
```
c = 0.4773
y = g(u)
x2 = f(u) + c * (g(u) - truth)
```

[30]:
```
np.round(np.mean(x2), 4)
```

[30]: np.float64(0.6932)

[26]:
```
se2 = x2.std() / np.sqrt(n)
```

[19]:
```
se2
```

[19]: np.float64(0.0002006231759532942)

[ ]:
```

```

0.2 Naive Monte Carlo in a BS World

[ ]:
```
import time
t1 = time.time()
```

[ ]:
```
def VanillaCallPayoff(spot, strike):
    return np.maximum(spot - strike, 0.0)
```

[ ]:
```
# The same old same old parameters

S = 41.0
K = 40.0
r = 0.08
v = 0.30
q = 0.0
T = 1.0
M = 10000 # number of MC replications
N = 252   # number of MC steps in a particular path
```

[ ]:
```
dt = T
nudt = (r - q - 0.5 * v * v) * dt
sigdt = v * np.sqrt(dt)
```

[ ]:
```
spot_t = np.empty((N))
call_t = np.empty(M)

z = np.random.normal(size=(M,N))

for i in range(M):
    spot_t[0] = S
```

```
    for j in range(1,N):
        spot_t[j] = S * np.exp(nudt + sigdt * z[i,j])
    call_t[i] = VanillaCallPayoff(spot_t[-1], K)
```

[ ]:
```
call_prc = np.exp(-r * T) * call_t.mean()
t2 = time.time()
```

[ ]:
```
call_prc
```

[ ]: 6.9288468728032111

[ ]:
```
se = call_t.std() / np.sqrt(M)
```

[ ]:
```
se
```

[ ]: 0.10848490988044222

[ ]:
```
print("The Naive Monte Carlo Price is: {0:.3f}".format(call_prc))
print("The Naive Monte Carlo StdErr is: {0:.6f}".format(se))
print("The total time take: {0}".format(t2-t1))
```

```
The Naive Monte Carlo Price is: 6.929
The Naive Monte Carlo StdErr is: 0.108485
The total time take: 4.806628227233887
```

[ ]:
```

```

0.2.1 The Control Variate Approach in a BS World

We will use the BS-Delta formula for our control variate. We can write the BS Delta function as follows:

[ ]:
```
from scipy.stats import norm
```

[ ]:
```
def BlackScholesDelta(spot, t, strike, expiry, volatility, rate, dividend):
    tau = expiry - t
    d1 = (np.log(spot/strike) + (rate - dividend + 0.5 * volatility *␣
 ↪volatility) * tau) / (volatility * np.sqrt(tau))
    delta = np.exp(-dividend * tau) * norm.cdf(d1)
    return delta
```

[ ]:
```
erddt = np.exp((r - q) * dt)
beta = -1.0

spot_t = np.empty((N))
call_t = np.empty(M)
#cash_flow_t = np.zeros((engine.replications, ))
z = np.random.normal(size=(M,N))
```

```
for i in range(M):
        #spot_t = spot
        convar = 0.0
        #z = np.random.normal(size=int(engine.time_steps))
        spot_t[0] = S

        for j in range(1,N):
            t = i * dt
            delta = BlackScholesDelta(S, t, K, T, v, r, q)
            spot_t[j] = spot_t[j-1] * np.exp(nudt + sigdt * z[i,j])
            convar += delta * (spot_t[j] -  spot_t[j-1]* erddt)
            #spot_t = spot_tn

        call_t[i] = VanillaCallPayoff(spot_t[-1], K) + beta * convar
```

```
/home/brough/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:3:
RuntimeWarning: divide by zero encountered in double_scalars
  app.launch_new_instance()
/home/brough/anaconda3/lib/python3.5/site-packages/ipykernel/__main__.py:3:
RuntimeWarning: invalid value encountered in sqrt
  app.launch_new_instance()
```

[ ]:
```
disc = np.exp(-r * T)
call_prc = disc * call_t.mean()
```

[ ]:
```
call_prc
```

[ ]:
```
nan
```

[ ]:
```

```