Simulation HW IV

Ze Yang (zey@andrew.cmu.edu)

February 14, 2018

```
In [1]: from abc import ABCMeta, abstractmethod
        import time
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        plt.style.use('ggplot')
        plt.rc('text', usetex=True)
        plt.rc('font', family='serif', size=15)
        %matplotlib inline

        import scipy.stats as stats
        from scipy.stats import norm
        from progressbar import ProgressBar
```

```
In [2]: def bs(t, S, K, T, sigma, r, div=0):
            tau = T - t
            rexp, dexp = np.exp(-r*tau), np.exp(-div*tau)
            d1 = (np.log(S/K) + (r-div+0.5*sigma**2)*tau) / (
                sigma*np.sqrt(tau))
            d2 = d1 - sigma*np.sqrt(tau)
            call = S*dexp*norm.cdf(d1) - K*rexp*norm.cdf(d2)
            put = K*rexp - S*dexp + call
            return call, put
```

```
In [135]: # Generators
          def gbm_exact(S0, sigma, r, div, T, n_steps):
              dt = T/n_steps
              S, i = S0, 0
              Z = np.random.normal(size=n_steps)
              while i < n_steps:
                  S *= np.exp((r-div-0.5*sigma**2)*dt + (
                      sigma*np.sqrt(dt)*Z[i]))
                  yield S
                  i += 1

          def hull_white_euler(S0, vv0, r, alpha,
                               psi, T, n_steps):
```

```
        dt = T/n_steps
        S, vv, i = S0, vv0, 0
        Z = np.random.normal(size=(2,n_steps))
        Z1, Z2 = Z[0], Z[1]
        while i < n_steps:
            S += r*S*dt + np.sqrt(vv)*S*np.sqrt(dt)*Z1[i]
            vv += alpha*vv*dt + psi*vv*np.sqrt(dt)*Z2[i]
            yield (S, vv)
            i += 1

    def CIR_exact(r0, alpha, sigma, b,
                  T, n_steps):
        dt = T/n_steps
        r, i = r0, 0
        d = int(np.round(
            (4*b*alpha)/(sigma)**2, 1))
        aexp = np.exp(-alpha*dt)
        while i < n_steps:
            lam = ((4*alpha*aexp)/(
                sigma**2*(1-aexp)))*r
            r = np.random.noncentral_chisquare(
                df=d, nonc=lam)*(
                sigma**2*(1-aexp))/(4*alpha)
            yield r
            i += 1
```

1 Hull-White Stochastic Volatility

1.1 Standard MC

```
In [107]: def hull_white_call(S0, K, T, r, vv0, alpha, psi,
                              n_steps, n_size, conditional=False):
              sample = np.zeros(n_size)
              bar = ProgressBar()
              for j in bar(range(n_size)):
                  S, vv = zip(*hull_white_euler(
                      S0, vv0, r, alpha, psi, T, n_steps))
                  if conditional:
                      sigma_hat = np.sqrt((1/n_steps)*sum(vv))
                      sample[j] = bs(0, S0, K, T, sigma_hat, r)[0]
                  else:
                      sample[j] = np.exp(-r*T)*np.clip(
                          S[-1]-K, 0, None)
              sample_mean = np.mean(sample)
              se = stats.sem(sample, ddof=0)
              return sample, sample_mean, se
```

```
In [149]: table = []
          for i, (alpha, psi) in enumerate(
              [(0.1,0.1),(0.1,1.0)]):
              _, price, se = hull_white_call(
                  S0=100, K=100, T=1, r=0.05, vv0=0.04,
                  alpha=alpha, psi=psi,
                  n_steps=50, n_size=10000)
              table.append([alpha, psi, price, se])
              print('''
              Hull-White call (alpha, psi)={}:
              Price = {}, SE = {}\n'''.format(
                  (alpha, psi), price, se))
              time.sleep(0.5)
```

```
100% (10000 of 10000) |###################| Elapsed Time: 0:00:03 Time: 0:00:03



    Hull-White call (alpha, psi)=(0.1, 0.1):
    Price = 10.734247140993515, SE = 0.1514908126407839



100% (10000 of 10000) |###################| Elapsed Time: 0:00:03 Time: 0:00:03



    Hull-White call (alpha, psi)=(0.1, 1.0):
    Price = 10.402796231918252, SE = 0.15512624052900922
```

1.2 Conditional MC

```
In [150]: for i, (alpha, psi) in enumerate(
              [(0.1,0.1),(0.1,1.0)]):
              _, price, se = hull_white_call(
                  S0=100, K=100, T=1, r=0.05, vv0=0.04,
                  alpha=alpha, psi=psi,
                  n_steps=50, n_size=10000,
                  conditional=1)
              table.append([alpha, psi, price, se])
              print('''
              Hull-White call (alpha, psi)={}:
              Price = {}, SE = {}\n'''.format(
                  (alpha, psi), price, se))
              time.sleep(0.5)
```

```
100% (10000 of 10000) |###################| Elapsed Time: 0:00:06 Time: 0:00:06
```

```
Hull-White call (alpha, psi)=(0.1, 0.1):
Price = 10.642658225571418, SE = 0.0022560351639518035
```

```
100% (10000 of 10000) |##################| Elapsed Time: 0:00:06 Time: 0:00:06
```

```
Hull-White call (alpha, psi)=(0.1, 1.0):
Price = 10.34079923768526, SE = 0.022448135433629095
```

In [151]:
```
index = [['Standard MC',
          'Conditional MC'], [1,2]]
index = pd.MultiIndex.from_product(index, names=['Method', 'Case'])
summary = pd.DataFrame(table, columns=[
    'Alpha', 'Psi', 'Price', 'SE'], index=index)
summary
```

Out[151]:

Method	Case	Alpha	Psi	Price	SE
Standard MC	1	0.1	0.1	10.734247	0.151491
	2	0.1	1.0	10.402796	0.155126
Conditional MC	1	0.1	0.1	10.642658	0.002256
	2	0.1	1.0	10.340799	0.022448

Conditonal MC reduced the standard error significantly.

2 Cox-Ingersol-Ross Spot Rate

In [108]:
```
def CIR_zcb(r0, alpha, sigma, b, T, n_steps, n_size):
    dt = T/n_steps
    sample = np.zeros(n_size)
    bar = ProgressBar()
    for j in bar(range(n_size)):
        r = list(CIR_exact(
            r0, alpha, sigma, b, T, n_steps))
        sample[j] = np.exp(-sum(r)*dt)
    sample_mean = np.mean(sample)
    se = stats.sem(sample, ddof=0)
    return sample, sample_mean, se

def CIR_caplet(r0, alpha, sigma, b, T,
               R, L, t_interval,
               n_steps, n_size):
```

```
        dt = T/n_steps
        sample = np.zeros(n_size)
        bar = ProgressBar()
        for j in bar(range(n_size)):
            r = list(CIR_exact(
                r0, alpha, sigma, b, T, n_steps))
            sample[j] = np.exp(-sum(r)*dt)*(
                L*t_interval*np.clip(r[-1]-R, 0, None))
        sample_mean = np.mean(sample)
        se = stats.sem(sample, ddof=0)
        return sample, sample_mean, se
```

```
In [109]: _, price, se = CIR_zcb(
              r0=0.04, alpha=0.2, sigma=0.1,
              b=0.05, T=1, n_steps=50, n_size=1000)
          print('''
          CIR zero coupon bond $1 face:
          Price = {}, SE = {}\n'''.format(price, se))
```

```
100% (1000 of 1000) |####################| Elapsed Time: 0:00:00 Time: 0:00:00



CIR zero coupon bond $1 face:
Price = 0.9597814284625382, SE = 0.0003347686556656842
```

```
In [110]: _, price, se = CIR_caplet(
              r0=0.04, alpha=0.2, sigma=0.1,
              b=0.05, T=1, R=0.05, L=1, t_interval=1/12,
              n_steps=50, n_size=1000)
          print('''
          CIR caplet R={}, ${} face:
          Price = {}, SE = {}\n'''.format(
              0.05, 1, price, se))
```

```
100% (1000 of 1000) |####################| Elapsed Time: 0:00:00 Time: 0:00:00



CIR caplet R=0.05, $1 face:
Price = 0.00034296941553407155, SE = 2.3634114685710555e-05
```

3 Greeks

3.1 Resimulation

In [133]:
```
def gbm_delta_resimulate(h, S0, sigma, r, div,
                         T, n_steps):
    dt = T/n_steps
    S_left, S_right, i = S0-h, S0+h, 0
    Z = np.random.normal(size=n_steps)
    while i < n_steps:
        transition = np.exp((r-div-0.5*sigma**2)*dt + (
            sigma*np.sqrt(dt)*Z[i]))
        S_left *= transition
        S_right *= transition
        yield S_left, S_right
        i += 1

def gbm_vega_resimulate(h, S0, sigma, r, div,
                        T, n_steps):
    dt = T/n_steps
    S_left, S_right, i = S0, S0, 0
    Z = np.random.normal(size=n_steps)
    while i < n_steps:
        S_left *= np.exp((r-div-0.5*(sigma-h)**2)*dt + (
            (sigma-h)*np.sqrt(dt)*Z[i]))
        S_right *= np.exp((r-div-0.5*(sigma+h)**2)*dt + (
            (sigma+h)*np.sqrt(dt)*Z[i]))
        yield S_left, S_right
        i += 1

def gbm_greeks(S0, K, sigma, r, div, T,
               h, path_sampler,
               n_steps, n_size, control_flag=False):
    sample_left =  np.zeros(n_size)
    sample_right = np.zeros(n_size)
    control =  np.zeros(n_size)
    bar = ProgressBar()
    for j in bar(range(n_size)):
        S_left, S_right = zip(*path_sampler(
            h, S0, sigma, r, div, T, n_steps))
        if control_flag:
            control[j] = S_left[-1]
        sample_left[j] = np.exp(-r*T)*np.clip(
            S_left[-1]-K, 0, None)
        sample_right[j] = np.exp(-r*T)*np.clip(
            S_right[-1]-K, 0, None)
    sample = (sample_right - sample_left) / (2*h)
    sample_mean = np.mean(sample)
```

```
    se = stats.sem(sample, ddof=0)
    if control_flag:
        gbm_mean = (S0 - h)*np.exp(r*T)
        adj = np.mean(control) - gbm_mean
        cov_xy = np.cov(control, sample)
        rho = np.corrcoef(control, sample)[0,1]
        a_hat = -cov_xy[0,1]/cov_xy[0,0]
        sample_mean += a_hat * adj
        se *= np.sqrt(1-rho**2)
    return sample, sample_mean, se
```

```
In [152]: table = []
          _, delta, se = gbm_greeks(
              S0=90, K=100, sigma=0.25, r=0.1,
              div=0.03, T=0.2, h=0.0001,
              path_sampler=gbm_delta_resimulate,
              n_steps=50, n_size=10000, control_flag=False)
          print('''
          GBM Resimulation:
          Delta = {}, SE of Delta = {}\n'''.format(delta, se))

          time.sleep(0.5)
          _, vega, se2 = gbm_greeks(
              S0=90, K=100, sigma=0.25, r=0.1,
              div=0.03, T=0.2, h=0.0001,
              path_sampler=gbm_vega_resimulate,
              n_steps=50, n_size=10000, control_flag=False)
          print('''
          GBM Resimulation:
          Vega = {}, SE of Vega = {}\n'''.format(vega, se2))
          table.append([delta, se, vega, se2])

          time.sleep(0.5)
          _, delta, se = gbm_greeks(
              S0=90, K=100, sigma=0.25, r=0.1,
              div=0.03, T=0.2, h=0.0001,
              path_sampler=gbm_delta_resimulate,
              n_steps=50, n_size=10000, control_flag=True)
          print('''
          GBM Resimulation (ST controled):
          Delta = {}, SE of Delta = {}\n'''.format(delta, se))

          time.sleep(0.5)
          _, vega, se2 = gbm_greeks(
              S0=90, K=100, sigma=0.25, r=0.1,
              div=0.03, T=0.2, h=0.0001,
              path_sampler=gbm_vega_resimulate,
              n_steps=50, n_size=10000, control_flag=True)
```

```
        print('''
        GBM Resimulation (ST controled):
        Vega = {}, SE of Vega = {}\n'''.format(vega, se2))
        table.append([delta, se, vega, se2])
```

```
100% (10000 of 10000) |###################| Elapsed Time: 0:00:02 Time: 0:00:02



GBM Resimulation:
Delta = 0.2153722921014939, SE of Delta = 0.0045162819695291935



100% (10000 of 10000) |###################| Elapsed Time: 0:00:04 Time: 0:00:04



GBM Resimulation:
Vega = 12.314588641577494, SE of Vega = 0.2767002225871702



100% (10000 of 10000) |###################| Elapsed Time: 0:00:02 Time: 0:00:02



GBM Resimulation (ST controled):
Delta = 0.23942766585446856, SE of Delta = 0.0030853030698220294



100% (10000 of 10000) |###################| Elapsed Time: 0:00:04 Time: 0:00:04



GBM Resimulation (ST controled):
Vega = 12.886542090902854, SE of Vega = 0.1775815365533063
```

3.2 Path Differentiation

```
In [142]: def gbm_greeks_pathdiff(S0, K, sigma, r, div, T,
                                  which, n_steps, n_size,
                                  control_flag=False):
              sample = np.zeros(n_size)
              control =  np.zeros(n_size)
              bar = ProgressBar()
```

```
        for j in bar(range(n_size)):
            S = list(gbm_exact(
                S0, sigma, r, div, T, n_steps))
            if control_flag:
                control[j] = S[-1]
            if which == 'delta':
                sample[j] = np.exp(-r*T)*(
                    S[-1]>=K)*(S[-1]/S0)
            elif which == 'vega':
                sample[j] = np.exp(-r*T)*(
                    S[-1]>=K)*S[-1]*(
                    (np.log(S[-1]/S0) - (
                        r-div+0.5*sigma**2)*T)/sigma)
            else:
                raise ValueError
        sample_mean = np.mean(sample)
        se = stats.sem(sample, ddof=0)
        if control_flag:
            gbm_mean = S0*np.exp(r*T)
            adj = np.mean(control) - gbm_mean
            cov_xy = np.cov(control, sample)
            rho = np.corrcoef(control, sample)[0,1]
            a_hat = -cov_xy[0,1]/cov_xy[0,0]
            sample_mean += a_hat * adj
            se *= np.sqrt(1-rho**2)
        return sample, sample_mean, se
```

In [153]:
```
_, delta, se = gbm_greeks_pathdiff(
    S0=90, K=100, sigma=0.25, r=0.1,
    div=0.03, T=0.2, which='delta',
    n_steps=50, n_size=10000, control_flag=False)
print('''
GBM Path differentiation:
Delta = {}, SE of Delta = {}\n'''.format(delta, se))

time.sleep(0.5)
_, vega, se2 = gbm_greeks_pathdiff(
    S0=90, K=100, sigma=0.25, r=0.1,
    div=0.03, T=0.2, which='vega',
    n_steps=50, n_size=10000, control_flag=False)
print('''
GBM Path differentiation:
Vega = {}, SE of Vega = {}\n'''.format(vega, se2))
table.append([delta, se, vega, se2])

time.sleep(0.5)
_, delta, se = gbm_greeks_pathdiff(
    S0=90, K=100, sigma=0.25, r=0.1,
```

```python
            div=0.03, T=0.2, which='delta',
            n_steps=50, n_size=10000, control_flag=True)
    print('''
    GBM Path differentiation (ST controled):
    Delta = {}, SE of Delta = {}\n'''.format(delta, se))

    time.sleep(0.5)
    _, vega, se2 = gbm_greeks_pathdiff(
        S0=90, K=100, sigma=0.25, r=0.1,
        div=0.03, T=0.2, which='vega',
        n_steps=50, n_size=10000, control_flag=True)
    print('''
    GBM Path differentiation (ST controled):
    Vega = {}, SE of Vega = {}\n'''.format(vega, se2))
    table.append([delta, se, vega, se2])
```

```
100% (10000 of 10000) |###################| Elapsed Time: 0:00:02 Time: 0:00:02



GBM Path differentiation:
Delta = 0.2215238344558542, SE of Delta = 0.00456715730452974



100% (10000 of 10000) |###################| Elapsed Time: 0:00:02 Time: 0:00:02



GBM Path differentiation:
Vega = 12.104970562784805, SE of Vega = 0.2753984598423407



100% (10000 of 10000) |###################| Elapsed Time: 0:00:02 Time: 0:00:02



GBM Path differentiation (ST controled):
Delta = 0.2416324230078789, SE of Delta = 0.0031031353006737007



100% (10000 of 10000) |###################| Elapsed Time: 0:00:02 Time: 0:00:02



GBM Path differentiation (ST controled):
Vega = 13.104778263032978, SE of Vega = 0.17803143209935848
```

3.3 Likelihood Estimate

```
In [146]: def gbm_greeks_likelihood(S0, K, sigma, r, div, T,
                                    which, n_steps, n_size,
                                    control_flag=False):
              sample = np.zeros(n_size)
              control =  np.zeros(n_size)
              bar = ProgressBar()
              for j in bar(range(n_size)):
                  S = list(gbm_exact(
                      S0, sigma, r, div, T, n_steps))
                  if control_flag:
                      control[j] = S[-1]
                  if which == 'delta':
                      sample[j] = np.exp(-r*T)*np.clip(
                      S[-1]-K, 0, None)*(1/(S0*T*sigma**2))*(
                          np.log(S[-1]/S0) - (r-div-0.5*sigma**2)*T)
                  elif which == 'vega':
                      y = np.log(S[-1]/S0) - (r-div-0.5*sigma**2)*T
                      z = sigma*np.sqrt(T)
                      sample[j] = np.exp(-r*T)*np.clip(
                          S[-1]-K, 0, None)*(
                          (-y/z)*(np.sqrt(T)*(1-(y/z**2)))-(1/sigma))
                  else:
                      raise ValueError
              sample_mean = np.mean(sample)
              se = stats.sem(sample, ddof=0)
              if control_flag:
                  gbm_mean = S0*np.exp(r*T)
                  adj = np.mean(control) - gbm_mean
                  cov_xy = np.cov(control, sample)
                  rho = np.corrcoef(control, sample)[0,1]
                  a_hat = -cov_xy[0,1]/cov_xy[0,0]
                  sample_mean += a_hat * adj
                  se *= np.sqrt(1-rho**2)
              return sample, sample_mean, se

In [154]: _, delta, se = gbm_greeks_likelihood(
              S0=90, K=100, sigma=0.25, r=0.1,
              div=0.03, T=0.2, which='delta',
              n_steps=50, n_size=10000, control_flag=False)
          print('''
          GBM likelihood estimation:
          Delta = {}, SE of Delta = {}\n'''.format(delta, se))

          time.sleep(0.5)
          _, vega, se2 = gbm_greeks_likelihood(
              S0=90, K=100, sigma=0.25, r=0.1,
              div=0.03, T=0.2, which='vega',
```

```python
            n_steps=50, n_size=10000, control_flag=False)
        print('''
        GBM likelihood estimation:
        Vega = {}, SE of Vega = {}\n'''.format(vega, se2))
        table.append([delta, se, vega, se2])

        time.sleep(0.5)
        _, delta, se = gbm_greeks_likelihood(
            S0=90, K=100, sigma=0.25, r=0.1,
            div=0.03, T=0.2, which='delta',
            n_steps=50, n_size=10000, control_flag=True)
        print('''
        GBM likelihood estimation (ST controled):
        Delta = {}, SE of Delta = {}\n'''.format(delta, se))

        time.sleep(0.5)
        _, vega, se2 = gbm_greeks_likelihood(
            S0=90, K=100, sigma=0.25, r=0.1,
            div=0.03, T=0.2, which='vega',
            n_steps=50, n_size=10000, control_flag=True)
        print('''
        GBM likelihood estimation (ST controled):
        Vega = {}, SE of Vega = {}\n'''.format(vega, se2))
        table.append([delta, se, vega, se2])
```

```
100% (10000 of 10000) |###################| Elapsed Time: 0:00:02 Time: 0:00:02



GBM likelihood estimation:
Delta = 0.2273725394231722, SE of Delta = 0.007677848658382844



100% (10000 of 10000) |###################| Elapsed Time: 0:00:02 Time: 0:00:02



GBM likelihood estimation:
Vega = 11.814197762328172, SE of Vega = 0.6799915646821565



100% (10000 of 10000) |###################| Elapsed Time: 0:00:02 Time: 0:00:02



GBM likelihood estimation (ST controled):
Delta = 0.25839497092116337, SE of Delta = 0.006541064887133898
```

```
100% (10000 of 10000) |###################| Elapsed Time: 0:00:02 Time: 0:00:02
```

```
GBM likelihood estimation (ST controled):
Vega = 14.328297824478517, SE of Vega = 0.6752024623006516
```

In [155]:
```python
index = [['Resimulation estimate',
          'Pathwise estimate',
          'Likelihood estimate'], ['None', 'ST']]
index = pd.MultiIndex.from_product(
    index, names=['Method', 'Control'])
summary = pd.DataFrame(table, columns=[
    'Delta Est', 'Delta Std Err', 'Vega Est', 'Vega Std Err'], index=index)
summary
```

Out[155]:

Method	Control	Delta Est	Delta Std Err	Vega Est
Resimulation estimate	None	0.215372	0.004516	12.314589
	ST	0.239428	0.003085	12.886542
Pathwise estimate	None	0.221524	0.004567	12.104971
	ST	0.241632	0.003103	13.104778
Likelihood estimate	None	0.227373	0.007678	11.814198
	ST	0.258395	0.006541	14.328298

Method	Control	Vega Std Err
Resimulation estimate	None	0.276700
	ST	0.177582
Pathwise estimate	None	0.275398
	ST	0.178031
Likelihood estimate	None	0.679992
	ST	0.675202

4 Digital Option Delta

4.1 Closed-form Solution

In last homework, we have shown that the price of a digital call is given by

$$c(t,x) = e^{-r(T-t)}N(d_2)$$

Where

$$d_2 = \frac{\log\frac{x}{K} + (r - \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}}; \qquad S_t = x$$

Therefore

$$\begin{aligned}\Delta &= e^{-r(T-t)}\frac{\partial}{\partial x}N(d_2)\\ &= e^{-r(T-t)}\frac{N'(d_2)}{\sigma x\sqrt{T-t}}\end{aligned}$$

```
In [159]: # For this problem we have
          d2 = (np.log(95/100) + (.05-.5*0.2**2)) / (0.2)
          delta = np.exp(-0.05)*norm.pdf(d2)/(0.2*95)
          print("The Black-Scholes Delta is:", delta)
```

```
The Black-Scholes Delta is: 0.019860050706
```

4.2 Resimulation Method

```
In [170]: def gbm_digital_delta(S0, K, sigma, r, div,
                                T, h, path_sampler,
                                n_steps, n_size):
              sample_left =  np.zeros(n_size)
              sample_right = np.zeros(n_size)
              control =  np.zeros(n_size)
              bar = ProgressBar()
              for j in bar(range(n_size)):
                  S_left, S_right = zip(*path_sampler(
                      h, S0, sigma, r, div, T, n_steps))
                  #if j % 10 == 0:print(S_left[-1], S_right[-1])
                  sample_left[j] = np.exp(-r*T)*(S_left[-1]>=K)
                  sample_right[j] = np.exp(-r*T)*(S_right[-1]>=K)
              sample = (sample_right - sample_left) / (2*h)
              sample_mean = np.mean(sample)
              se = stats.sem(sample, ddof=0)
              return sample, sample_mean, se

          sample, delta, se = gbm_digital_delta(
              S0=95, K=100, sigma=0.2, r=0.05,
              div=0, T=1, h=0.0001,
              path_sampler=gbm_delta_resimulate,
              n_steps=100, n_size=10000)
          print('''
          GBM Resimulation for Digital Option:
          Delta = {}, SE of Delta = {}\n'''.format(delta, se))
```

```
100% (10000 of 10000) |##################| Elapsed Time: 0:00:04 Time: 0:00:04



GBM Resimulation for Digital Option:
Delta = 0.0, SE of Delta = 0.0
```

The resimulation method performed very badly, because we are asking for a very small $h = 0.0001$, but only allow for a small sample size of $n = 10,000$. Consequently, it is highly unlikely that K lies between the final stock prices produced by the intial price pair $S_0 - h$ and $S_0 + h$.

We get a reasonable estimate using $h = 0.005$, and sample size $n = 300,000$, which took 2.5 minutes to run. So bad!

```
In [171]: sample, delta, se = gbm_digital_delta(
              S0=95, K=100, sigma=0.2, r=0.05,
              div=0, T=1, h=0.005,
              path_sampler=gbm_delta_resimulate,
              n_steps=100, n_size=300000)
          print('''
          GBM Resimulation for Digital Option:
          Delta = {}, SE of Delta = {}\n'''.format(delta, se))
```

```
100% (300000 of 300000) |#################| Elapsed Time: 0:02:21 Time: 0:02:21



GBM Resimulation for Digital Option:
Delta = 0.020927047339015712, SE of Delta = 0.002575658089607897
```

4.3 Likelihood Method

```
In [173]: def gbm_likelihood_digital_delta(S0, K, sigma, r, div,
                                           T, n_steps, n_size):
              sample = np.zeros(n_size)
              bar = ProgressBar()
              for j in bar(range(n_size)):
                  S = list(gbm_exact(
                      S0, sigma, r, div, T, n_steps))
                  sample[j] = np.exp(-r*T)*(S[-1]>=K)*(
                      1/(S0*T*sigma**2))*(
                      np.log(S[-1]/S0) - (r-div-0.5*sigma**2)*T)
              sample_mean = np.mean(sample)
              se = stats.sem(sample, ddof=0)
              return sample, sample_mean, se

          sample, delta, se = gbm_likelihood_digital_delta(
              S0=95, K=100, sigma=0.2, r=0.05,
              div=0, T=1, n_steps=100, n_size=10000)
          print('''
          GBM Likelihood Estimate for Digital Option:
          Delta = {}, SE of Delta = {}\n'''.format(delta, se))
```

```
100% (10000 of 10000) |###################| Elapsed Time: 0:00:04 Time: 0:00:04
```

```
GBM Likelihood Estimate for Digital Option:
Delta = 0.020046603571571883, SE of Delta = 0.0002959985626061557
```

Likelihood method is much better in this case. We simulated only 10000 samples, and the standard error is about 1/10 of the resimulation method estimate.

```
In [ ]:
```