finm320_24_hw6

May 10, 2024

[1]:
```python
import numpy as np
from  sympy import Matrix
from scipy.stats import norm
```

```
/opt/anaconda3/lib/python3.8/site-packages/scipy/__init__.py:146: UserWarning: A
NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy
(detected version 1.24.4
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

0.1 Problem 1

0.1.1 (a)

$$
\begin{aligned}
Cov(X_T) &= \mathbb{E}[\Sigma W_T W_T^T \Sigma^T] \\
&= \Sigma Cov(W_T)\Sigma^T \\
&= \begin{bmatrix} 0.3 & 0 \\ 0 & 0.2 \end{bmatrix} \begin{bmatrix} 1T & 0.8T \\ 0.8T & 1T \end{bmatrix} \begin{bmatrix} 0.3 & 0 \\ 0 & 0.2 \end{bmatrix} \\
&= \begin{bmatrix} 0.09T & 0.048T \\ 0.048T & 0.04T \end{bmatrix}
\end{aligned}
$$

[2]:
```python
vol1 = 0.3
vol2 = 0.2
corr = 0.8
sigma = Matrix([[vol1, 0],[0, vol2]])
cov_Wt = Matrix([[1, corr],[corr, 1]])
sigmaT = sigma.T

cov = sigma @ cov_Wt @ sigmaT
cov
```

[2]:
$$\begin{bmatrix} 0.09 & 0.048 \\ 0.048 & 0.04 \end{bmatrix}$$

0.1.2 (b)

[3]:
```python
class MultiGBM:

    def __init__(self,S0,r,correlations,sigma):
        self.S0 = S0
```

```
        self.r = r
        self.correlations = correlations
        self.sigma = sigma
```

[4]:
```
hw6p1dynamics = MultiGBM(S0=np.array([100,110]),r=0.05,
                         correlations = np.array([[1,0.8],[0.8,1]]),
                         sigma = np.diag([0.3, 0.2]))
```

[5]:
```
class CallOnBasket:

    def __init__(self,K,T,weights):
        self.K = K
        self.T = T
        self.weights = weights
```

[6]:
```
hw6p1contract=CallOnBasket(K=110,T=1.0,weights = np.array([1/2, 1/2]))
```

[7]:
```
class MCengine:

    def __init__(self, M, antithetic, control, seed):
        self.M = M                                  # How many simulations
        self.antithetic = antithetic
        self.control = control
        self.rng = np.random.default_rng(seed=seed) # Seeding the random number␣
↪generator with a specified number helps make the calculations reproducible

    def price_callonbasket_multiGBM(MC,contract,dynamics):

        # You complete the coding of this function.
        # self.rng.multivariate_normal may be useful.
        # See documentation for numpy.random.Generator.multivariate_normal
        # as self.rng is an instance of numpy.random.Generator

        # You are not required to support the case where MC.control = MC.
↪antithetic = True
        # (simultaneous use of control variate and antithetic)
        # But you are required to support the other 3 possible settings of MC.
↪antithetic/MC.control
        # namely False/False, True/False, False/True.
        # (ordinary MC, antithetic without control, control without antithetic)

        K, T, weights = contract.K, contract.T, contract.weights
        S0, r, corr, sigma = dynamics.S0, dynamics.r, dynamics.correlations,␣
↪dynamics.sigma
        M, antithetic, control = MC.M, MC.antithetic, MC.control

        X_0 = (np.ones((M, len(S0)))* np.log(S0)).T
```

```
        sigma_mat = (np.ones((M, len(S0))) * np.diag(sigma)).T

        dW_T = MC.rng.multivariate_normal([0,0],corr,M) * np.sqrt(T)
        X_T = X_0 + (r-sigma_mat**2/2)*T + sigma.dot(dW_T.T)

        S_T = np.exp(X_T)
        H_T = weights.dot(S_T)
        C_T = np.exp(-r*T) * np.maximum(H_T-K, 0)

        call_price = np.mean(C_T)
        standard_error = np.std(C_T, ddof=1) / np.sqrt(M)

        def BScallPrice(sigma,S,rGrow,r,K,T):
            F=S*np.exp(rGrow*T)
            sd = sigma*np.sqrt(T)
            d1 = np.log(F/K)/sd+sd/2
            d2 = d1-sd
            return np.exp(-r*T)*(F*norm.cdf(d1)-K*norm.cdf(d2))

        if control:
            sig = np.sqrt(sigma[0,0]**2 + sigma[1,1]**2 +␣
 ↪2*sigma[0,0]*sigma[1,1]*corr[0,1])/2
            S = (S0[0]*S0[1])**0.5
            rGrow = r + (2*sigma[0,0]*sigma[1,1]*corr[0,1] - sigma[0,0]**2 -␣
 ↪sigma[1,1]**2)/8

            C_BS = BScallPrice(sig,S,rGrow,r,K,T)

            G_Tcon = np.exp(np.mean(X_T, axis=0))
            C_Tcon = np.exp(-r*T) * np.maximum(G_Tcon-K, 0)

            cov = np.cov(C_T, C_Tcon)
            beta = cov[0,1] / cov[1,1]
            Y_con = C_T + beta*(C_BS - C_Tcon)
            call_price = np.mean(Y_con)
            standard_error = np.std(Y_con, ddof=1) / np.sqrt(M)

        return(call_price, standard_error)
```

[8]:
```
hw6p1bMC=MCengine(M=10000,antithetic=False,control=False,seed=0)
(call_price_ordinary, std_err_ordinary) = hw6p1bMC.
 ↪price_callonbasket_multiGBM(hw6p1contract,hw6p1dynamics)
print(call_price_ordinary, std_err_ordinary)
```

```
9.858103798706601 0.1680048813661487
```

0.1.3 (c)

[9]:
```
hw6p1cMC=MCengine(M=10000,antithetic=True,control=False,seed=0)
(call_price_AV, std_err_AV) = hw6p1cMC.
 ↪price_callonbasket_multiGBM(hw6p1contract,hw6p1dynamics)
print(call_price_AV, std_err_AV)
```

```
9.858103798706601 0.1680048813661487
```

0.1.4 (d)

$$\begin{aligned}\log G_t &= \frac{1}{2}\Bigg(\log S_t^{[1]} + \log S_t^{[2]}\Bigg)\\ &= \frac{1}{2}\Bigg(\log S_0^{[1]} + (r - \frac{\sigma_{[1]}^2}{2})t + \sigma_{[1]}W_t^{[1]} + \log S_0^{[2]} + (r - \frac{\sigma_{[2]}^2}{2})t + \sigma_{[2]}W_t^{[2]}\Bigg)\\ &= \frac{1}{2}\Bigg(\log{(S_0^{[1]}S_0^{[2]})} + (2r - \frac{\sigma_{[1]}^2 + \sigma_{[2]}^2}{2})t + \sigma_{[1]}W_t^{[1]} + \sigma_{[2]}W_t^{[2]}\Bigg)\end{aligned}$$

Therefore

$$\mathbb{E}[\log G_T] = \frac{1}{2}\Bigg(\log{(S_0^{[1]}S_0^{[2]})} + (2 * r - \frac{\sigma_{[1]}^2 + \sigma_{[2]}^2}{2})T\Bigg)$$

$$\begin{aligned}\mathrm{Var}[\log G_T] &= \mathrm{Var}\left[\frac{1}{2}\left(\log S_T^{[1]} + \log S_T^{[2]}\right)\right]\\ &= \frac{1}{4}\mathrm{Var}\left(\log S_T^{[1]} + \log S_T^{[2]}\right)\\ &= \frac{1}{4}\left[\mathrm{Var}\left(\log S_T^{[1]}\right) + \mathrm{Var}\left(\log S_T^{[2]}\right) + 2\mathrm{Cov}\left(\log S_T^{[1]}, \log S_T^{[2]}\right)\right]\\ &= \frac{\sigma_{[1]}^2 + 2\rho\sigma_{[1]}\sigma_{[2]} + \sigma_{[2]}^2}{4}T\end{aligned}$$

0.1.5 (e)

$$C^G = C^{BS}((S_0^{[1]}S_0^{[2]})^{\frac{1}{2}}, 0, K, T, r + \frac{2\sigma_{[1]}\sigma_{[2]}\rho_{[1],[2]} - \sigma_{[1]}^2 - \sigma_{[2]}^2}{8}, r, \frac{\sqrt{\sigma_{[1]}^2 + \sigma_{[2]}^2 + 2\sigma_{[1]}\sigma_{[2]}\rho_{[1],[2]}}}{2})$$

0.1.6 (f)

[10]:
```
hw6p1fMC=MCengine(M=10000,antithetic=False,control=True,seed=0)
(call_price_CV, std_err_CV) = hw6p1fMC.
 ↪price_callonbasket_multiGBM(hw6p1contract,hw6p1dynamics)
print(call_price_CV, std_err_CV)
```

```
9.993510290823446 0.004473729471333335
```

0.2 Problem 2

0.2.1 (a)

[11]:
```
class GBM:

    def __init__(self,sigma,r,S0):
        self.sigma = sigma
        self.r = r
        self.S0 = S0
```

[12]:
```
hw6p2dynamics=GBM(sigma=0.2,r=0.02,S0=100)
```

[13]:
```
class CallOption:

    def __init__(self,K,T):
        self.K=K
        self.T=T
```

[14]:
```
hw6p2contract=CallOption(K=150,T=1)
```

[15]:
```
class MCimportanceEngine:

    def __init__(self, M, lamb, seed):
        self.M = M                                    # How many simulations
        self.lamb = lamb                              # drift adjustment
        self.rng = np.random.default_rng(seed=seed) # Seeding the random number␣
↪generator with a specified number helps make the calculations reproducible

    def price_call_GBM(self, contract,dynamics):

        # You complete the coding of this function.
        # self.rng.normal may be useful.
        # See documentation for numpy.random.Generator.normal
        # as self.rng is an instance of numpy.random.Generator

        K, T = contract.K, contract.T
        S0, r, sigma = dynamics.S0, dynamics.r, dynamics.sigma
        M, lamb = self.M, self.lamb

        W_1 = self.rng.normal(0,T,M)
        X_T = np.log(S0) + (r-sigma**2/2)*T + sigma*(W_1 + lamb*T)
        S_T = np.exp(X_T)
        Y_T = np.exp(-r*T - lamb*W_1 - 0.5*T*lamb**2) * np.maximum(S_T-K, 0)

        call_price = np.mean(Y_T)
        standard_error = np.std(Y_T, ddof=1) / np.sqrt(M)
```

```
        return(call_price, standard_error)
```

[16]:
```
hw6p2aMC=MCimportanceEngine(M=100000,lamb=0,seed=0) #zero drift adjustment␣
 ↪gives ordinary MC

(call_price_ordinary, std_err_ordinary) =  hw6p2aMC.
 ↪price_call_GBM(hw6p2contract,hw6p2dynamics)
print(call_price_ordinary, std_err_ordinary)
```

```
0.25270332833609405 0.007609293292996182
```

0.2.2 (b)

Change the drift in BM:

$$
\begin{aligned}
d\tilde{W}_t &= dW_t^* + \lambda dt \\
dS_t &= rS_t dt + \sigma S_t d\tilde{W}_t = rS_t dt + \sigma S_t (dW_t^* + \lambda t) = (r + \sigma\lambda) S_t dt + \sigma S_t dW_t^* \\
d\log S_t &= (r + \sigma\lambda - \frac{\sigma^2}{2})dt + \sigma dW_t^* \\
\mathbb{E}^*[S_t] &= S_0 e^{(r+\sigma\lambda)t} \\
\lambda &= \frac{\ln(\frac{\mathbb{E}^*[S_t]}{S_0})/t - r}{\sigma} \\
&= (\ln(\frac{165}{100}) - 0.02)/0.2 = 2.404
\end{aligned}
$$

0.2.3 (c)

[17]:
```
# Calculate Lamb
ES_T = 165
S_0 = hw6p2dynamics.S0
r = hw6p2dynamics.r
sigma = hw6p2dynamics.sigma
T = hw6p2contract.T

lamb_value= (np.log(ES_T/S_0)/T - r) / sigma
```

[18]:
```
hw6p2cMC=MCimportanceEngine(M=100000,lamb=lamb_value,seed=0) # Fill in the lamb␣
 ↪parameter with the lambda that you compute in (b)
(call_price_importsamp, std_err_importsamp) =  hw6p2cMC.
 ↪price_call_GBM(hw6p2contract,hw6p2dynamics)
print(call_price_importsamp, std_err_importsamp)
```

```
0.24843662621391502 0.0007734271968138013
```

[ ]:
```

```