FIM 548-001 HW-3

Yi-Siou Feng, Raiden Han, Tingyu Lei

March 23, 2022

Perface

All codes in this report are Python codes. To implement these methods, we need to import the NumPy package initially. Also, we include two statistic packages for calculation.

```python
import numpy as np
import statsmodels.api as sm
from scipy.stats import norm

np.random.seed(323)
```

Problem 1

(1) First, we need to generate the correlated normal-distributed random variables. From the Cholosky decomposition, we generated $Z_i \overset{iid}{\sim} N(0,1), i = 1,2,3$, and let

$$\begin{aligned}
X_1 &= Z_1, \\
X_2 &= \rho_{1,2} Z_1 + \sqrt{1-\rho_{1,2}^2} Z_2, \\
X_3 &= \rho_{1,3} Z_1 + \frac{\rho_{2,3} - \rho_{1,3}\rho_{1,2}}{\sqrt{1-\rho_{1,2}^2}} Z_2 + \sqrt{1-\rho_{1,3}^2 - \frac{(\rho_{2,3}-\rho_{1,3}\rho_{1,2})^2}{1-\rho_{1,2}^2}} Z_3.
\end{aligned}$$

Then, the stock prices at time T are

$$S_T^i = S_0^i \exp\left(\left(r - \delta_i - \sigma_i^2/2\right) T + \sigma\sqrt{T} X_i\right), \quad i = 1,2,3.$$

From the stock prices, we can derive the current fair price of the option

$$V = e^{-rT}\left(S_T^1 + S_T^2 - S_T^3 - K\right)^+.$$

We repeated the above experiment $N = 10^8$ times with the antithetic variate method to reduce the variance. The option price estimate is 20.40, and the estimator's variance is 2.8621×10^{-6}.

(2) Using the same idea, we can derive the stock prices sequence

$$S_{t_j}^i = S_{t_{j-1}}^i \exp\left(\left(r - \delta_i - \sigma_i^2/2\right) \Delta t_j + \sigma\sqrt{\Delta t_j} X_j^i\right), \quad i = 1,2,3;\ j = 1,\ldots,12,$$

where $\Delta t_j = t_j - t_{j-1}$, (X_j^1, X_j^2, X_j^3) is a set of random numbers as described before. Based on the least-squares Monte-Carlo method, we can work backward to derive the fair price of the option. Since the terminal values of the option price can be directly derived, for $j = 11$ to 1, using $N = 10^8$ samples, we can run a regression among all samples

$$V_{t_{j+1}}^n = \boldsymbol{\beta\phi}\left(\boldsymbol{S_{t_j}^n}\right),$$

where $V_{t_{j+1}}^n$ is the option's price at time t_{j+1} for sample n, $\boldsymbol{S_{t_j}^n} = (S_{t_j}^{n,1}, S_{t_j}^{n,2}, S_{t_j}^{n,3})$, ie the stock prices at time t_j for sample n. In this solution, we use 22 basis functions (including the constant):

$$\begin{aligned}
\phi_1\left(S_{t_j}^1, S_{t_j}^2, S_{t_j}^3\right) &= 1, \\
\phi_2\left(S_{t_j}^1, S_{t_j}^2, S_{t_j}^3\right) &= S_{t_j}^1, \\
\phi_3\left(S_{t_j}^1, S_{t_j}^2, S_{t_j}^3\right) &= S_{t_j}^2, \\
\phi_4\left(S_{t_j}^1, S_{t_j}^2, S_{t_j}^3\right) &= S_{t_j}^3, \\
\phi_5\left(S_{t_j}^1, S_{t_j}^2, S_{t_j}^3\right) &= \left(S_{t_j}^1\right)^2, \\
\phi_6\left(S_{t_j}^1, S_{t_j}^2, S_{t_j}^3\right) &= \left(S_{t_j}^2\right)^2, \\
\phi_7\left(S_{t_j}^1, S_{t_j}^2, S_{t_j}^3\right) &= \left(S_{t_j}^3\right)^2, \\
\phi_8\left(S_{t_j}^1, S_{t_j}^2, S_{t_j}^3\right) &= S_{t_j}^1 S_{t_j}^2, \\
\phi_9\left(S_{t_j}^1, S_{t_j}^2, S_{t_j}^3\right) &= S_{t_j}^1 S_{t_j}^3, \\
\phi_{10}\left(S_{t_j}^1, S_{t_j}^2, S_{t_j}^3\right) &= S_{t_j}^2 S_{t_j}^3, \\
\phi_{11}\left(S_{t_j}^1, S_{t_j}^2, S_{t_j}^3\right) &= \left(S_{t_j}^1\right)^3, \\
\phi_{12}\left(S_{t_j}^1, S_{t_j}^2, S_{t_j}^3\right) &= \left(S_{t_j}^2\right)^3, \\
\phi_{13}\left(S_{t_j}^1, S_{t_j}^2, S_{t_j}^3\right) &= \left(S_{t_j}^3\right)^3, \\
\phi_{14}\left(S_{t_j}^1, S_{t_j}^2, S_{t_j}^3\right) &= \left(S_{t_j}^1\right)^2 S_{t_j}^2, \\
\phi_{15}\left(S_{t_j}^1, S_{t_j}^2, S_{t_j}^3\right) &= \left(S_{t_j}^1\right)^2 S_{t_j}^3, \\
\phi_{16}\left(S_{t_j}^1, S_{t_j}^2, S_{t_j}^3\right) &= \left(S_{t_j}^2\right)^2 S_{t_j}^1, \\
\phi_{17}\left(S_{t_j}^1, S_{t_j}^2, S_{t_j}^3\right) &= \left(S_{t_j}^2\right)^2 S_{t_j}^3, \\
\phi_{18}\left(S_{t_j}^1, S_{t_j}^2, S_{t_j}^3\right) &= \left(S_{t_j}^3\right)^2 S_{t_j}^1, \\
\phi_{19}\left(S_{t_j}^1, S_{t_j}^2, S_{t_j}^3\right) &= \left(S_{t_j}^3\right)^2 S_{t_j}^2, \\
\phi_{20}\left(S_{t_j}^1, S_{t_j}^2, S_{t_j}^3\right) &= S_{t_j}^1 S_{t_j}^2 S_{t_j}^3, \\
\phi_{21}\left(S_{t_j}^1, S_{t_j}^2, S_{t_j}^3\right) &= \max\left\{S_{t_j}^1, S_{t_j}^2, S_{t_j}^3\right\}, \\
\phi_{22}\left(S_{t_j}^1, S_{t_j}^2, S_{t_j}^3\right) &= \left(\max\left\{S_{t_j}^1, S_{t_j}^2, S_{t_j}^3\right\} - K\right)^+,
\end{aligned}$$

indicating $\boldsymbol{\phi}\left(\boldsymbol{S_{t_j}^n}\right) = \left(\phi_1\left(\boldsymbol{S_{t_j}^n}\right), \ldots, \phi_{22}\left(\boldsymbol{S_{t_j}^n}\right)\right)$. Using these estimated coefficients, we can compare the expected holding value

$$EV_{t_{j+1}}^n = e^{-r\Delta t_{j+1}} \hat{\boldsymbol{\beta}} \boldsymbol{\phi}\left(\boldsymbol{S_{t_j}^n}\right)$$

and the intrinsic value

$$h\left(\boldsymbol{S_{t_j}^n}\right) = \left(S_{t_j}^{n,1} + S_{t_j}^{n,2} - S_{t_j}^{n,3} - K\right)^+,$$

and let $V_{t_j}^n$ take the higher one. Finally, the current option price estimate is

$$V_0 = e^{-r\Delta t_1} \frac{1}{N} \sum_{n=1}^{N} V_{t_1}^n.$$

From our simulation, when $N = 10^8$ with the antithetic variate method, the option price estimate is 20.15, and the estimator's variance is 1.5966×10^{-6}. The value is lower than the one from Problem 1.1, which violates the boundary of an American option. This is because our pricing is based on regressions, and model predictions may result in early exercise. However, if we view the three stocks as one non-dividend asset, the option looks like a vanilla American call option. In theory, it is never optimal to early exercise an American call option on a non-dividend asset. Therefore, the price is naturally lower due to the introduction of early exercise.

Problem 2

Since we are required to estimate the 99% 10-day VaR, we can simulate the stock price in 10 days first. Again, using the Cholosky decomposition, for each simulation, we can generate $Z_1, Z_2 \overset{iid}{\sim} N(0,1)$, and

$$\begin{aligned} X_1 &= Z_1, \\ X_2 &= \rho Z_1 + \sqrt{1-\rho^2} Z_2. \end{aligned}$$

Let $t = 10/252$, the stock prices in 10 days are

$$S_t^i = S_0^i \exp\left(\left(r - \sigma_i^2/2\right) t + \sigma_i \sqrt{t} X_i\right).$$

For the delta method and the delta-gamma method, we can derive the initial delta and gamma for both options. The estimated value change of the portfolio is

$$\Delta V^{delta} = \sum_{i=1}^{2} \Delta_i \left(S_t^i - S_0^i\right),$$

$$\Delta V^{gamma} = \sum_{i=1}^{2} \left[\Delta_i \left(S_t^i - S_0^i\right) + \frac{1}{2} \Gamma_i \left(S_t^i - S_0^i\right)^2\right].$$

Because this is a short position in this portfolio, the 99% 10-day VaR is simply the left-side 99% quantile of the changed value sequence.

For the full simulation approach, we calculate the portfolio's current value, and simulate the stocks' prices in ten days. Assume all the other parameters are the same, we can use the Black-Scholes formula to compute the portfolio's value in ten days. Again, we take the value difference of these two time points, and calculate the VaR.

Based on $N = 10^8$ simulations with antithetic variate method, the estimated 10-day VaR from the delta method is 3.87, and the one from the delta-gamma method is 4.70, and it is 4.40 from the full simulation approach.

Problem 3

Since the asset value fits a Geometric Brownian motion, the minimum value conditional on the terminal asset value is a precise estimate. So there is no need to discretize the interval. In other words, we choose $h = T$.

Usually, we can generate the geometric Brownian motion from

$$V_T = V_0 \exp\left(\left(\mu - \sigma^2/2\right) T + \sigma\sqrt{T}X\right).$$

If we plug the expression in the given formula, we have

$$\begin{aligned}
M_i &= \exp\left\{\frac{1}{2}\left(\log\left(V_T V_0\right) - \sqrt{\left[\log\left(V_T/V_0\right)\right]^2 - 2\sigma^2 T\log(U)}\right)\right\} \\
&= \exp\left\{\frac{1}{2}\left(\log\left(V_0^2\exp\left(\left(\mu - \sigma^2/2\right) T + \sigma\sqrt{T}X\right)\right) - \right.\right. \\
&\quad \left.\left.\sqrt{\left[\log\left(\exp\left(\left(\mu - \sigma^2/2\right) T + \sigma\sqrt{T}X\right)\right)\right]^2 - 2\sigma^2 T\log(U)}\right)\right\} \\
&= S_0\exp\left\{\frac{1}{2}\left[\left(r - \frac{\sigma^2}{2}\right)T + \sigma\sqrt{T}X - \sqrt{\left[\left(r - \frac{\sigma^2}{2}\right)T + \sigma\sqrt{T}X\right]^2 - 2\sigma^2 T\log(U)}\right]\right\}.
\end{aligned}$$

To compute the probability of default before time T, in each simulation, we generate $X \sim N(0, 1)$ and $U \sim Unif(0, 1)$. Then, we use the above formula to find the simulated minimum value M using the formula given. Accordingly, by repeating the experiment $N = 10^8$ times with the antithetic variate method, we are able to get a sequence of M. Lastly, we calculate the ratio of M less than the default boundary B, which is the desired probability of default, and the value is 76.34%. An important note is that, because the relationship between X and U are analytically complex, we simply use $-X$ and new iid U for antithetic samples.

Appendix - Python Code

April 7, 2022

[1]:
```python
import numpy as np
import statsmodels.api as sm
from scipy.stats import norm

np.random.seed(323)
```

[2]:
```python
def reinforce_function(func, n, *args):
    """ Repeat a function with certain parameters several times and return the
    mean result

    Parameters
    ----------
    func : function
        The function to be repeated
    n : int
        The repeating time
    args : tuple
        The parameters for the function

    Returns
    -------
    mean_value : float
        The mean value of n times running
    """

    value = []
    for i in range(n):
        value.append(func(*args))
    mean_value = np.mean(value, axis=0)

    return mean_value


def reinforce_series_function(func, n, *args):
    """ Repeat a function, which generates a series of results, with certain
    parameters several times and return the estimator's value and variance

    Parameters
```

```
    ----------
    func : function
        The function to be repeated
    n : int
        The repeating time
    args : tuple
        The parameters for the function

    Returns
    -------
    mean : float
        The mean value of all the results
    variance : float
        The variance of the estimator
    """

    value_list = []
    for i in range(n):
        value_list.append(func(*args))
    value = np.hstack(value_list)
    mean = np.mean(value)
    variance = np.var(value) / value.shape[0]

    return mean, variance
```

1 Problem 1

[3]:
```
def euro_basket_option(n):
    """

    Parameters
    ----------
    n : int or float
        The simulation size

    Returns
    -------
    v_red : ndarray
        The simulated present values

    """

    # Convert the simulation size into an integer
    n = int(n)
    # Initialize parameters of the contract and assets
    r, delta, sigma, rho = 0.05, 0.02, 0.3, 0.2
```

```python
    s0 = np.full((3, 1), 100)
    k, t = 100, 1
    # Define parameters for the multivariate normal distribution
    mean, cov = np.zeros(3), np.full((3, 3), rho)
    np.fill_diagonal(cov, 1)
    rng = np.random.default_rng()
    # Generate multivariate normal random variables
    x = rng.multivariate_normal(mean, cov, size=n).T
    # Use the antithetic method to reduce the variance
    x = np.hstack((x, -x))
    # Calculate terminal prices
    st = s0 * np.exp((r - delta - sigma ** 2 / 2) * t + sigma * np.sqrt(t) * x)
    # Return present values of the contract
    v = np.exp(-r * t) * np.maximum(st[0, :] + st[1, :] - st[2, :] - k, 0)
    v_red = (v[:n] + v[n:]) / 2

    return v_red
```

[4]:
```python
# Output the results
euro_basket_price, euro_basket_var = reinforce_series_function(
    euro_basket_option, 10, 1e7)
print('European Basket Option:')
print('Price: {:.2f}'.format(euro_basket_price))
print('Variance of the Estimator: {:.4e}'.format(euro_basket_var))
```

```
European Basket Option:
Price: 20.40
Variance of the Estimator: 2.8921e-06
```

[5]:
```python
def berm_basket_option(n):
    """

    Parameters
    ----------
    n : int or float
        The simulation size

    Returns
    -------
    ev : float
        The present value of the contract
    se2 : float
        The variance of the estimator

    """

    # Convert the simulation size into an integer
    n = int(n)
```

```python
# Initialize parameters of the contract and assets
r, delta, sigma, rho = 0.05, 0.02, 0.3, 0.2
k, t, m = 100, 1, 12
h = t / m
s = np.empty(shape=(3, 2 * n, m + 1))
s[:, :, 0] = 100
# Define parameters for the multivariate normal distribution
mean, cov = np.zeros(3), np.full((3, 3), rho)
np.fill_diagonal(cov, 1)
rng = np.random.default_rng()
# Generate multivariate normal random variables
x = rng.multivariate_normal(mean, cov, size=(m, n)).T
# Use the antithetic method to reduce the variance
x = np.hstack((x, -x))
# Generate price paths for all simulations
for i in range(m):
    s[:, :, i + 1] = s[:, :, i] * np.exp(
        (r - delta - sigma ** 2 / 2) * h + sigma * np.sqrt(h) * x[:, :, i])
# Initialize containers
v = np.empty(shape=(m, 2 * n))
cv = np.empty(shape=(m - 1, 2 * n))

# Define a function to calculate the intrinsic value
def cal_intrin_value(s, k, i):
    return np.maximum(s[0, :, i] + s[1, :, i] - s[2, :, i] - k, 0)

# Calculate the value of the option at time t = T
v[m - 1, :] = np.maximum(cal_intrin_value(s, k, m), 0)
# Backward induction
for i in range(m - 1, -1, -1):
    # Calculate basis functions' values
    X = np.vstack([s[j, :, i] for j in range(3)] +
                  [s[j, :, i] * s[p, :, i]
                   for j in range(3) for p in range(j, 3)] +
                  [s[j, :, i] * s[p, :, i] * s[q, :, i] for j in range(3)
                   for p in range(j, 3) for q in range(p, 3)] +
                  [np.max(s[:, :, i], axis=0)] +
                  [np.maximum(np.max(s[:, :, i], axis=0) - k, 0)])
    X = sm.add_constant(X.T)
    # Create and fit the OLS model
    model = sm.OLS(v[i, :], X)
    results = model.fit()
    # Using the coefficients, calculate the expected value
    cv[i - 1, :] = np.maximum(np.exp(-r * h) * results.fittedvalues, 0)
    # Set the option's value as its discounted value
    v[i - 1, :] = np.exp(-r * h) * v[i, :]
    # If the option's intrinsic value is greater than its expected value,
```

```python
        # set its value to its intrinsic value
        intrin_value = cal_intrin_value(s, k, i)
        ind = cv[i - 1, :] < intrin_value
        v[i - 1, ind] = intrin_value[ind]
    # Return present values of the contract
    pv = np.exp(-r * h) * v[0, :]
    pv_red = (pv[:n] + pv[n:]) / 2

    return pv_red
```

[6]:
```python
# Output the results
berm_basket_price, berm_basket_var = reinforce_series_function(
    berm_basket_option, 100, 1e6)
print('Bermudan Basket Option:')
print('Price: {:.2f}'.format(berm_basket_price))
print('Variance of the Estimator: {:.4e}'.format(berm_basket_var))
```

```
Bermudan Basket Option:
Price: 20.15
Variance of the Estimator: 1.5966e-06
```

2 Problem 2

[7]:
```python
def bs_formula_d1(st, k, r, sigma, t):
    """ Calculate the d1 value in the Black-Scholes model

    Parameters
    ----------
    st : float
        The spot price of the asset
    k : float
        The strike price
    r : float
        The risk-free rate
    sigma : float
        The volatility of the asset
    t : float
        The time to maturity

    Returns
    -------
    d1 : float
        The d1 value in the Black-Scholes model

    """

    d1 = (np.log(st / k) + (r + sigma ** 2 / 2) * t) / (sigma * np.sqrt(t))
```

```
    return d1


def bs_formula(st, k, r, sigma, t, type):
    """Use the Black-Scholes model to calculate the option's price

    Parameters
    ----------
    st : float
        The spot price of the asset
    k : float
        The strike price
    r : float
        The risk-free rate
    sigma : float
        The volatility of the asset
    t : float
        The time to maturity
    type : {'call', 'put'}
        The option type

    Returns
    -------
    v : float
        The option price

    """
    d1 = bs_formula_d1(st, k, r, sigma, t)
    d2 = d1 - sigma * np.sqrt(t)
    if type == 'call':
        v = norm.cdf(d1) * st - norm.cdf(d2) * k * np.exp(-r * t)
    elif type == 'put':
        v = norm.cdf(-d2) * k * np.exp(-r * t) - norm.cdf(-d1) * st

    return v


def d10_99var_delta_gamma_full(n):
    """Calculate the 10-day 99% VaR using the three methods

    Parameters
    ----------
    n : int or float
        The simulation size

    Returns
```

```
    -------
    d10_99var_delta : float
        The estimated 10-day 99% VaR using the delta method
    d10_99var_gamma : float
        The estimated 10-day 99% VaR using the delta-gamma method
    d10_99var_full : float
        The actual 10-day 99% VaR based on the simulation

    """

    # Convert the simulation size into an integer
    n = int(n)
    # Initialize parameters of options and assets
    sc0, kc, volc, mc = 50, 51, 0.28, 0.75
    sp0, kp, volp, mp = 20, 19, 0.25, 1
    r, rho, t = 0.06, 0.4, 10 / 252
    # Using the Black-Scholes formula, calculate d1 values for both options
    dc1 = bs_formula_d1(sc0, kc, r, volc, mc)
    dp1 = bs_formula_d1(sp0, kp, r, volp, mp)
    # Calculate both options' delta's and gamma's
    delta_c = norm.cdf(dc1)
    delta_p = norm.cdf(dp1) - 1
    gamma_c = norm.pdf(dc1) / (sc0 * volc * np.sqrt(mc))
    gamma_p = norm.pdf(dp1) / (sp0 * volp * np.sqrt(mp))
    # Data vectorization
    s0 = np.array([sc0, sp0]).reshape((2, -1))
    vol = np.array([volc, volp]).reshape((2, -1))
    delta = np.array([delta_c, delta_p]).reshape((2, -1))
    gamma = np.array([gamma_c, gamma_p]).reshape((2, -1))
    # Define parameters for the multivariate normal distribution
    mean = np.zeros(2)
    corr = np.full((2, 2), rho)
    np.fill_diagonal(corr, 1)
    # Generate multivariate normal random variables
    rng = np.random.default_rng()
    x = rng.multivariate_normal(mean, corr, size=n).T
    # Use the antithetic method to reduce the variance
    x = np.hstack((x, -x))
    # Calculate stock prices in 10 days
    st = s0 * (np.exp(
        (r - vol ** 2 / 2) * t + vol * np.sqrt(t) * x))
    # Use the delta method to estimate the 10-day 99% VaR
    ds = st - s0
    dv_delta = delta.T @ ds
    # Use the delta-gamma method to estimate
    dv_gamma = delta.T @ ds + gamma.T @ ds ** 2 / 2
    # Calculate fair prices of both options using the full simulation
```

```python
    v = bs_formula(sc0, kc, r, volc, mc, 'call') + bs_formula(
        sp0, kp, r, volp, mp, 'put')
    vt = bs_formula(st[0, :], kc, r, volc, mc - t, 'call') + bs_formula(
        st[1, :], kp, r, volp, mp - t, 'put')
    dv = vt - v
    # Calculate the 10-day 99% VaR
    d10_99var_delta = np.quantile(dv_delta, .99)
    d10_99var_gamma = np.quantile(dv_gamma, .99)
    d10_99var_full = np.quantile(dv, .99)

    return d10_99var_delta, d10_99var_gamma, d10_99var_full
```

[8]:
```python
# Output the results
delta_es, gamma_es, full_es = reinforce_function(
    d10_99var_delta_gamma_full, 10, 1e7)
print('Estimated 10-day 99% VaR:')
print('Delta Method: {:.2f}'.format(delta_es))
print('Delta-Gamma Method: {:.2f}'.format(gamma_es))
print('Full Simulation Approach: {:.2f}'.format(full_es))
```

```
Estimated 10-day 99% VaR:
Delta Method: 3.87
Delta-Gamma Method: 4.70
Full Simulation Approach: 4.40
```

3 Problem 3

[9]:
```python
def cal_default_prob(n):
    """

    Parameters
    ----------
    n : int or float
        The simulation size

    Returns
    -------
    default_prob : float
        The probability of default

    """

    # Convert the simulation size into an integer
    n = int(n)
    # Initialize parameters of the asset and the contract
    v0, mu, sigma, b, t = 100, 0.03, 0.4, 70, 5
    # Generate normal random variables
```

```python
    x = np.random.normal(size=n)
    # Use the antithetic method to reduce the variance
    x = np.hstack((x, -x))
    # Generate uniform random variables
    u = np.random.uniform(size=2 * n)
    # Using the property of the Brownian bridge, calculate minima
    inc_rate = (mu - sigma ** 2 / 2) * t + sigma * np.sqrt(t) * x
    m = np.exp((2 * np.log(v0) + inc_rate - np.sqrt(
        inc_rate ** 2 - 2 * sigma ** 2 * t * np.log(u))) / 2)
    # Derive the default probability
    default_prob = (m <= b).sum() / (2 * n)

    return default_prob
```

[10]:
```python
# Output the result
prob_d = reinforce_function(cal_default_prob, 10, 1e5)
print('Probability of Default: {:.2%}'.format(prob_d))
```

```
Probability of Default: 76.34%
```