FIM 548-001 HW-2

Yi-Siou Feng, Raiden Han, Tingyu Lei

February 4, 2022

Perface

All codes in this report are Python codes. To implement these methods, we need to import the NumPy package initially.

```
import numpy as np
np.random.seed(204)
```

Problem 1

Since X_1, X_2 and X_3 are standard normal variables with correlation $\rho_{1,2}$, $\rho_{1,3}$ and $\rho_{2,3}$, the covariance matrix can be written as

$$\boldsymbol{\Sigma} = \begin{pmatrix} 1 & \rho_{1,2} & \rho_{1,3} \\ \rho_{1,2} & 1 & \rho_{2,3} \\ \rho_{1,3} & \rho_{2,3} & 1 \end{pmatrix}.$$

Using Cholosky decomposition, we have $\boldsymbol{\Sigma} = \boldsymbol{A}\boldsymbol{A}'$, where $\boldsymbol{A}$ is a lower triangular matrix, and the elements in it can be calculated as

$$a_{11} = \sqrt{\boldsymbol{\Sigma}_{11}} = 1,$$

$$a_{21} = \frac{1}{a_{11}}\left(\boldsymbol{\Sigma}_{21}\right) = \frac{1}{1}\rho_{1,2} = \rho_{1,2},$$

$$a_{31} = \frac{1}{a_{11}}\left(\boldsymbol{\Sigma}_{31}\right) = \frac{1}{1}\rho_{1,3} = \rho_{1,3},$$

$$a_{22} = \sqrt{\boldsymbol{\Sigma}_{22} - a_{21}^2} = \sqrt{1-\rho_{1,2}^2},$$

$$a_{32} = \frac{1}{a_{22}}\left(\boldsymbol{\Sigma}_{32} - a_{31}a_{21}\right) = \frac{1}{\sqrt{1-\rho_{1,2}^2}}\left(\rho_{2,3} - \rho_{1,3}\rho_{1,2}\right),$$

$$a_{33} = \sqrt{\boldsymbol{\Sigma}_{33} - a_{31}^2 - a_{32}^2} = \sqrt{1-\rho_{1,3}^2 - \left(\frac{\rho_{2,3}-\rho_{1,3}\rho_{1,2}}{\sqrt{1-\rho_{1,2}^2}}\right)^2}.$$

Therefore,

$$\boldsymbol{A} = \begin{pmatrix} 1 & 0 & 0 \\ \rho_{1,2} & \sqrt{1-\rho_{1,2}^2} & 0 \\ \rho_{1,3} & \frac{\rho_{2,3}-\rho_{1,3}\rho_{1,2}}{\sqrt{1-\rho_{1,2}^2}} & \sqrt{1-\rho_{1,3}^2-\frac{(\rho_{2,3}-\rho_{1,3}\rho_{1,2})^2}{1-\rho_{1,2}^2}} \end{pmatrix}.$$

To simulate the multivariate normal distribution, for each sample, we first simulate three independent standard normal random variables Z_1, Z_2 and Z_3. Then, let

$$X_1 = Z_1,$$

$$X_2 = \rho_{1,2}Z_1 + \sqrt{1-\rho_{1,2}^2}Z_2,$$

$$X_3 = \rho_{1,3}Z_1 + \frac{\rho_{2,3}-\rho_{1,3}\rho_{1,2}}{\sqrt{1-\rho_{1,2}^2}}Z_2 + \sqrt{1-\rho_{1,3}^2-\frac{(\rho_{2,3}-\rho_{1,3}\rho_{1,2})^2}{1-\rho_{1,2}^2}}Z_3,$$

and (X_1, X_2, X_3) is from the desired multivariate normal distribution.

Problem 2

The payoff structure of the contract can be written as

$$V(c) = \begin{cases} \mathbb{E}\left[\sum_{i=1}^{m} e^{-rt_i}c_i + e^{-rT}S_T\right] & ,\tau \geq T, S_T \leq K, \\ \mathbb{E}\left[\sum_{i=1}^{m} e^{-rt_i}c_i + e^{-rT}S_0\right] & ,\tau \geq T, S_T > K, \\ \mathbb{E}\left[\sum_{i=1}^{i^\tau} e^{-rt_i}c_i + e^{-r\tau}S_0\right] & ,\tau < T, \end{cases}$$

where

$$i^\tau = \max\left\{i : i_0 \leq i \leq m, i \in \mathbb{N}, t_i \leq \tau\right\}.$$

Since all parameters except c are given in the question, we can generate $N = 100,000,000$ stock price samples fitting a geometric Brownian motion and manually adjust c until

$$\frac{1}{N}\sum_{i=1}^{N} V_i(c) \approx 100.00.$$

The code is shown in the Appendix, and we assure the accuracy by running the code many times. It turned out that the result is always 100.00 when $c = 5.64\%$ and $N = 100,000,000$.

Problem 3

Using the Euler scheme, we can simulate the stock price under the local volatility model, and price the European call option. Specifically, we segment the entire lifetime of the option into m intervals and denote the endpoints as $t_0, t_1, \ldots, t_m$. Denote $\Delta t = \frac{T}{m}$. For $i = 0, 1, \ldots, m-1$,

$$\sigma_{t_i} = \frac{1}{2e^{t_i}}\left(\frac{100}{S_{t_i}}\right)^{0.3},$$

and the stock price is

$$S_{t_{i+1}} = (1 + r\Delta t)S_{t_i} + \sigma_{t_i} S_{t_i} \sqrt{\Delta t} Z_i,$$

where $Z_i \overset{iid}{\sim} N(0,1)$. Repeating the process for N times, we numerically derive the option price as

$$c \approx \frac{1}{N} \sum_{i=1}^{N} e^{-rT} (S_T^i - K)^+.$$

The method for determining accuracy is the same as described in Problem 2 and will not be repeated below. For space efficiency reasons, we segment the samples into 10 batches, with 1,000,000 samples each batch. In total, we simulate $N = 10,000,000$ price samples in $m = 252$ steps, and the present value of the option is $c = 11.0$.

Problem 4

Similar to Problem 3, we use the Euler scheme again. For each simulation, we generate two independent standard normal random variables Z_1 and Z_2, and we use the Cholesky decomposition to generate two standard normal random variables with correlation ρ.

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

One more time, we segment the sample into m_{total} intervals. Given $S_0 = 100$, $v_0 = 0.04$, the whole path can be generated by

$$\begin{aligned} v(t_i) &= v(t_{i-1}) + \kappa(\theta - v^+(t_{i-1}))\Delta t + \sigma \sqrt{v^+(t_{i-1})} \sqrt{\Delta t} Z_v, \\ S(t_i) &= S(t_{i-1}) \exp\left(\left(r - \frac{v(t_{i-1})}{2} \right) \Delta t + \sqrt{v^+(t_{i-1})} \sqrt{\Delta t} Z_s \right), \end{aligned}$$

where $i = 1, 2, \ldots, m_{total}$. After N simulations, the value of the lookback call option is

$$c_m = \frac{1}{n} \sum_{i=1}^{n} e^{-rT} \left(\max_{1 \le j \le m, j \in \mathbb{N}} S_{t_j}^i - K \right)^+.$$

We take total step number as $m_{total} = 252$, which means we simulate the stock price half trading day once. Also, 252 is a multiple of 12, making it easier for our observation. By generating $N = 1,000,000$ samples for each m, we identify the fair prices of these options for $m = 3, 6, 12$ are

$$\begin{aligned} c_3 &= 0.3, \\ c_6 &= 0.3, \\ c_{12} &= 0.3. \end{aligned}$$

Problem 5

We run N loops and calculate the mean to price the reset strike option. First, we have

$$\omega = \frac{1}{\nu} \log\left(1 - \theta\nu - \frac{\sigma^2 \nu}{2} \right).$$

For each loop, we simulate stock prices at equally-divided interval endpoints $t_1, \ldots, t_m$. Denote $\Delta t = t_i - t_{i-1}$,

$$X_{t_i} = \theta G_i + \sigma\sqrt{G_i} Z_i,$$
$$S_{t_i} = S_{t_{i-1}} e^{(r-\delta+\omega)\Delta t + X_{t_i}},$$

where $i = 1, \ldots, m$, $Z_i \overset{iid}{\sim} N(0,1)$, $G_i \overset{iid}{\sim} \text{Gamma}(h/\nu, \nu)$. Finally, the fair price of the call option is

$$c = \frac{1}{n} \sum_{i=1}^{n} e^{-rT_2} \left(S^i_{T_2} - \min\left(K, S^i_{T_1}\right)\right)^+ .$$

For each sample, we calculate the stock price for half an hour. So for the option with a maturity $T_2 = 0.5$, it contains 126 trading days. Given that the regular trading hours are from 9:30 a.m. to 4:00 p.m., we have 13 prices each trading day, which means there are $m = 126 \times 13 = 1638$ steps in each simulation. In total, we generate $N = 10,000,000$ samples in 100 batches and calculate the option's price. The present value of the reset strike option is

$$c = 7.11.$$

Appendix - Python Code

February 28, 2022

[1]:
```
import numpy as np

np.random.seed(204)
```

1 Problem 2

[2]:
```
def arcc_price(c, n):
    """ Calculate the value of an autocallable reverse convertible contract
    using the Monte-Carlo method

    Parameters
    ----------
    c : float
        The annual coupon
    n : int
        The number of simulations

    Returns
    -------
    ev : float
        The simulated value of the contract

    """

    # Initialize the parameters of the contract
    s0, r, sigma, k, t, m, i0 = 100, 0.04, 0.3, 55, 1, 4, 2
    # Calculate the coupon at each payment
    ci = c * t / m
    # Calculate the time interval
    h = t / m
    # Simulate the stock prices
    s_normal = np.random.normal(size=(m, n))
    stock_price = np.exp((r - sigma ** 2 / 2) * h +
                         sigma * np.sqrt(h) * s_normal)
    stock_price = np.cumprod(stock_price, axis=0) * s0
    # Calculate tau
    tau = stock_price[i0 - 1:, :] >= s0
```

```
    tau = np.where(tau.any(axis=0), tau.argmax(axis=0) + i0, m)
    # Calculate the no-arbitrage price of the contract
    # Calculate the coupon for different tau's
    coupon = np.array([np.exp(-r * i * h) * ci for i in range(1, m + 1)])
    coupon = np.cumsum(coupon)
    # Calculate the price for each simulation
    v = coupon[tau - 1] + np.where(
        tau < m, np.exp(-r * tau * h) * s0, np.exp(-r * t) * np.where(
            stock_price[-1, :] > k, s0, stock_price[-1, :]))
    # Calculate the mean
    ev = np.mean(v)

    return ev
```

[3]:
```
# Print the result
print('The contract price when c = 5.64%: {:.2f}'.format(
    arcc_price(5.64, int(1e8))))
```

```
The contract price when c = 5.64%: 100.00
```

2 Problem 3

[4]:
```
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)

    return mean_value
```

[5]:
```python
def loc_vol_option(step, n):
    """Calculate the option price when the stock price follows a local
    volatility model

    Parameters
    ----------
    step : int
        The number of steps to calculate the SDE in each simulation
    n : int
        The number of simulations

    Returns
    -------
    ec : float
        The fair value of the call option at present
    """

    # Initialize the parameters of the contract and the stock
    s0, k, r, t = 100, 110, 0.05, 1
    delta = t / step

    # Define the function for calculating volatility
    def sigma(st, t):
        return np.power((100 / st), 0.3) / (2 * np.exp(t))

    # Generate standard normal random variables
    z = np.random.normal(size=(n, step))
    # Initialize a container to store the price
    s = np.zeros(shape=(n, step + 1))
    s[:, 0] = s0
    # Use the Euler Scheme to simulate the SDE
    for i in range(step):
        s[:, i + 1] = (1 + r * delta) * s[:, i] + sigma(s[:, i], i * delta) * \
                      s[:, i] * np.sqrt(delta) * z[:, i]
    # Calculate the mean of expected values
    c = np.maximum(s[:, step] - k, 0)
    ec = np.mean(c) * np.exp(-r * t)

    return ec
```

[6]:
```python
# Print the result
print('Option Price: {:.1f}'.format(
    reinforce_function(loc_vol_option, 10, 252, int(1e6))))
```

```
Option Price: 11.0
```

3 Problem 4

[7]:
```
def lookback_call_option(m, step, n):
    """

    Parameters
    ----------
    m : int
        The number of time intervals for observation
    step : int
        The number of steps between two observation points in each simulation
    n : int
        The number of simulations

    Returns
    -------
    ec : float
        The fair value of the call option at present
    """

    # Initialize parameters for stock prices
    s0, v0, r, kappa, theta, sigma_v, rho = 100, 0.04, 0.03, 2, 0.04, 0.5, -0.7
    # Initialize parameters for the option
    t, k = 0.5, 120
    # Calculate total steps and time intervals
    n_step = m * step
    delta = t / n_step
    # Generate standard normal random variables
    z1, z2 = np.random.normal(size=(2, n, n_step))
    # Using Cholesky decomposition to generate correlated normal r.v.s
    zv, zs = z1, rho * z1 + np.sqrt(1 - rho ** 2) * z2
    # Initialize containers to store volatility's and prices
    v = np.zeros(shape=(n, n_step + 1))
    s = np.zeros(shape=(n, n_step + 1))
    v[:, 0] = v0
    s[:, 0] = s0
    # Use the Euler Scheme to simulate the SDE
    for i in range(n_step):
        v[:, i + 1] = v[:, i] + kappa * (theta - np.maximum(v[:, i], 0)) * \
            delta + sigma_v * np.sqrt(
            np.maximum(v[:, i], 0) * delta) * zv[:, i]
        s[:, i + 1] = s[:, i] * np.exp((r - v[:, i] / 2) * delta + np.sqrt(
            np.maximum(v[:, i], 0) * delta) * zs[:, i])
    # Calculate observation indices
    observation_index = [i * step for i in range(1, m + 1)]
    # Calculate the option's price
    c = np.max(np.maximum(s[:, observation_index] - k, 0), axis=1)
```

```
    ec = np.mean(c) * np.exp(-r * t)

    return ec
```

[8]:
```
# Print the result
print('Option Price when m = 3: {:.1f}'.format(
    lookback_call_option(3, 84, int(1e6))))
print('Option Price when m = 6: {:.1f}'.format(
    lookback_call_option(6, 42, int(1e6))))
print('Option Price when m = 12: {:.1f}'.format(
    lookback_call_option(12, 21, int(1e6))))
```

```
Option Price when m = 3: 0.3
Option Price when m = 6: 0.3
Option Price when m = 12: 0.3
```

4 Problem 5

[9]:
```
def reset_strike_option(step, n):
    """Calculate the reset strike option of call type price when the stock
    price follows a variance gamma process

    Parameters
    ----------
    step : int
        The number of steps to calculate the SDE in each simulation
    n : int
        The number of simulations

    Returns
    -------
    ec : float
        The fair value of the reset strike option at present
    """
    # Initialize parameters for stock prices
    theta, sigma, nu, s0, r, delta = -0.1, 0.2, 0.2, 100, 0.03, 0
    # Initialize parameters for the option
    k, t1, t2 = 100, 0.25, 0.5
    # Calculate the time interval and omega
    h = t2 / step
    omega = np.log(1 - theta * nu - sigma ** 2 * nu / 2) / nu
    # Generate standard normal random variables and gamma random variables
    z = np.random.normal(size=(n, step))
    g = np.random.gamma(shape=h / nu, scale=nu, size=(n, step))
    x = theta * g + sigma * np.sqrt(g) * z
    # Initialize a container to store stock prices
    s = np.zeros(shape=(n, step + 1))
```

```
    s[:, 0] = s0
    # Simulate the stock price
    for i in range(step):
        s[:, i + 1] = s[:, i] * np.exp((r - delta + omega) * h + x[:, i])
    # Calculate the reset date index
    reset_index = int(np.round(step * t1 / t2))
    # Calculate the mean of expected payoffs
    c = np.maximum(s[:, step] - np.minimum(k, s[:, reset_index]), 0)
    ec = np.mean(c) * np.exp(-r * t2)

    return ec
```

[10]:
```
# Print the result
print('Option Price: {:.2f}'.format(
    reinforce_function(reset_strike_option, 100, 1638, int(1e5))))
```

```
Option Price: 7.11
```