FIM 548-001 HW-4

Yi-Siou Feng, Raiden Han, Tingyu Lei

April 12, 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 GPU packages for acceleration.

```
import numpy as np
import cupy as cp
from cupyx.scipy.special import ndtr

np.random.seed(412)
```

Problem 1

First, we need to generate the correlated normal random variables. As described in the question,

$$(X_1, \ldots, X_{10}) \sim N(\mathbf{0}, \mathbf{\Sigma}),$$

where $\mathbf{\Sigma}$ is the covariance matrix with diagonal entries of 1 and all other entries of 0.2. We can use the Cholesky decomposition to derive the formulas from iid normal-distributed variables. For simplicity, we used the Python build-in function to generate. Under the assumption of exponential distribution with constant default intensity rates, bond i's default time τ_i is

$$\tau_i = F_i^{-1}(N(X_i)),$$

where $N(\cdot)$ is the cumulative probability function of the standard normal distribution, and

$$F_i(x) = 1 - e^{-\lambda_i x}.$$

Equivalently,

$$F_i^{-1}(u) = -\frac{\log(1-u)}{\lambda_i},$$

where λ_i is bond i's default intensity rate. Alternatively, we can also use the built-in function to implement. Then, we can manually adjust s so that the mean of $V(\tau_1, \ldots, \tau_{10})$ over $N = 10^8$ simulations is zero. For each simulation, we find the rank statistics $(\tau_{(1)}, \ldots, \tau_{(10)})$. Then, the value of the CDS is

$$V(\tau_1, \ldots, \tau_{10}) = V_{value}(\tau_1, \ldots, \tau_{10}) - V_{prot}(\tau_1, \ldots, \tau_{10}),$$

where

$$V_{value}\left(\tau_1,\ldots,\tau_{10}\right) = (1-R)\, e^{-r\tau_{(5)}} \mathbb{I}\left(\tau_{(5)} \le T\right)$$

with R the recovery rate of the fifth bond to default, and

$$V_{prot}\left(\tau_1,\ldots,\tau_{10}\right) = \begin{cases} \sum_{i=1}^{j} se^{-rT_i} + se^{-r\tau_{(5)}} \frac{\tau_{(5)}-T_j}{T_{j+1}-T_j}, & T_j \le \tau_{(5)} \le T_{j+1}, \\ \sum_{i=1}^{5} se^{-rT_i}, & \tau_{(5)} > T, \end{cases}$$

with $j = 0, 1, \ldots, 5$ and $T_j = j$. The fair annual protection payment rate is 1.42%.

Problem 2

The cumulative distribution function of exit time T is

$$F_T(t) = \int_0^t f(u)\, du = \int_0^t \frac{1}{18} u\, du = \frac{t^2}{36}, \quad 0 \le t \le 6.$$

Therefore, for $0 \le t \le 6$, its inverse function is

$$F_T^{-1}(u) = 6\sqrt{u}, \quad 0 \le u \le 1.$$

Using the formula above, we can simulate T from the inverse method. Assume $X(0)$ is given. In each simulation, we simulate $U \sim Unif(0,1)$ and $Z \sim N(0,1)$, and the exit time

$$T = 6\sqrt{U}.$$

So the asset price at exit time T is

$$X(T) = X(0)e^{\left(r-\sigma^2/2\right)T + \sigma\sqrt{T}Z}.$$

The outcome to Series B investor is

$$f(X(T)) = \begin{cases} X(T)\frac{I_B}{P_B}, & X(T) \ge 1,000, \\ \max\left\{\min\left\{\frac{I_B}{I_A+I_B}X(T), I_B\right\}, X(T)\frac{I_B}{P_B}\right\}, & X(T) < 1,000. \end{cases}$$

Again, we manually adjusted $X(0)$, such that the mean of $e^{-rT}f(X(T))$ over $N = 10^{10}$ simulations equals to I_B. The value of $X(0)$ is 672.5. And the overvaluation is

$$\frac{P_B - X(0)}{X(0)} = 48.70\%.$$

Appendix - Python Code

April 19, 2022

[1]:
```
import numpy as np
import cupy as cp
from cupyx.scipy.special import ndtr

cp.random.seed(412)
```

[2]:
```
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))
    value = cp.array(value)
    mean_value = cp.mean(value, axis=0).item()

    return mean_value
```

1 Problem 1

[3]:
```python
def basket_cds(n, s):
    """

    Parameters
    ----------
    n : int or float
        The simulation size
    s : float
        The annual protection payment for the CDS

    Returns
    -------
    v : float
        The simulated present value of the CDS

    """

    # Convert the simulation size into an integer
    n = int(n)
    # Initialize parameters of the contract and assets
    r, t = 0.03, 5
    default_int = cp.array([0.05, 0.01, 0.05, 0.05, 0.01, 0.1, 0.01, 0.09,
                            0.1, 0.02]).reshape(10, -1)
    recov_rate = cp.array([0.1, 0.1, 0.3, 0.1, 0.3, 0.1, 0.2, 0.2,
                           0.1, 0.1]).reshape(10, -1)
    # Define parameters for the multivariate normal distribution
    cov = cp.full((10, 10), 0.2)
    cp.fill_diagonal(cov, 1)
    # Use the Cholesky decomposition to derive the transformation matrix
    a = cp.linalg.cholesky(cov)
    # Generate the correlated normal random variables
    z = cp.random.normal(size=(10, n))
    x = cp.dot(a, z)
    # Use the antithetic method to reduce the variance
    x = cp.hstack((x, -x))
    # Calculate the default time
    tau = -cp.log(1 - ndtr(x)) / default_int.reshape((10, 1))
    # Select samples with the fifth default time less than 5 years
    tau_rank_ind = cp.argsort(tau, axis=0)
    tau_5 = tau[tau_rank_ind[4, :], range(2 * n)]
    tau_5_bool = tau_5 <= t
    # Calculate the possible income for CDS holders
    v_value = cp.zeros(2 * n)
    v_value[tau_5_bool] = ((1 - recov_rate[tau_rank_ind[4, tau_5_bool]]).T *
                           cp.exp(-r * tau_5[tau_5_bool])).ravel()
```

```python
    # Calculate the discounted payments
    payments = [s * cp.exp(-r * (i + 1)).get() for i in range(t)]
    cum_payments = cp.cumsum(cp.array([0] + payments))
    payments = cp.array(payments)
    # Find the last complete payment when the CDS is triggered
    frac_t, comp_t = cp.modf(tau_5[tau_5_bool])
    comp_t = comp_t.astype('int')
    # Calculate the outcome for CDS holders
    v_port = cp.full(2 * n, cp.sum(payments))
    v_port[tau_5_bool] = cum_payments[comp_t] + payments[comp_t] * frac_t * \
                         cp.exp(-r * tau_5[tau_5_bool])
    # Calculate the CDS's value
    v = v_value - v_port
    ev = cp.mean(v)

    return ev
```

[4]:
```python
# Output the results
print("The fair value of the CDS is ${:.4f}.".format(
    reinforce_function(basket_cds, 100, 1e6, .01421)))
```

```
The fair value of the CDS is $0.0000.
```

2 Problem 2

[5]:
```python
def startup_valuation(n, x0):
    # Convert the simulation size into an integer
    n = int(n)
    # Initialize parameters of the asset
    pa, pb, ia, ib = 450, 1000, 50, 100
    r, sigma = 0.025, 0.9
    # Generate random variables
    u = cp.random.uniform(size=n)
    t = 6 * cp.sqrt(u)
    z = cp.random.normal(size=n)
    # Calculate the exit value
    xt = x0 * cp.exp((r - sigma ** 2 / 2) * t + sigma * cp.sqrt(t) * z)
    # Calculate the payout to Series B investors
    outcome = cp.where(xt < 1000,
                       cp.maximum(cp.minimum((ib / (ia + ib)) * xt, ib),
                                  xt * ib / pb),
                       xt * ib / pb)
    eo = cp.mean(cp.exp(-r * t) * outcome).item()

    return eo
```

[6]:
```python
# Output the results
print("The fair investment for Series B investor is ${:.2f}.".format(
    reinforce_function(startup_valuation, 1000, 1e7, 672.52)))
print("The overvaluation is {:.2%}.".format((1000 - 672.5) / 672.5))
```

```
The fair investment for Series B investor is $100.00.
The overvaluation is 48.70%.
```