# COMP 7405 Assignment 3 Report

NG Ho Hin (3035754155)

## Interface

It is a Python command-line based interface which takes user input corresponding to a functionality in the project specification.

```
Mini Option Pricer
-------------------------------------
1. European Option
2. Implied Volatility Calculator
3. American Option
4. Geometric Asian Option
5. Arithmetic Asian Option
6. Geometric Basket Option
7. Arithmetic Basket Option
9. Exit
(Enter an option)
```

After selecting a function, user can follow with parametric input (every input in decimal), except for certain input being coerced as integer only (such as the ***n*** observation period for asian options). The input should follow the order in the prompt.

*Sample input for Option [1] European Option*

```
(Enter an option) 1
(Enter spot price, volatility, risk-free rate, repo rate, time to maturity (years), strike in decimals)
(Separated by a space) 50 0.223144 0.05 0 2 52
```

In this case, the corresponding input parsed would be spot = 50, volatility = 22.3144%, risk-free rate = 5%, repo rate = 0%, time to maturity = 2 years, strike = 52.

After entering options arguments, user will need to enter the option type (call/put) using an integer.

```
(Enter option type [1. call / 2. put]) 1
```

Some options allow further selection of sub-functions, such as for European option, we have choice of using closed-form formula to price or using binomial tree. Respective options would have their own arguments requirement if applicable. In the demo case here, if binomial tree is chosen, users will also have to enter the number of steps of the tree.

```
(Enter pricing method [1. closed-form / 2. binomial tree]) 2
(Enter steps) 10
```

The pricing results will be printed on the console, and user brought back to main menu. By default functions print the result to screen, but internally each function also return the option price, or a tuple containing all output (such as confident interval).

```
Spot Price Grid
[[ 50.        55.24711   61.044864  67.451046  74.529508  82.350799  90.992873 100.541866 111.092951 122.751291 135.633082]
 [  0.        45.251236  50.        55.24711   61.044864  67.451046  74.529508  82.350799  90.992873 100.541866 111.092951]
 [  0.         0.        40.953486  45.251236  50.        55.24711   61.044864  67.451046  74.529508  82.350799  90.992873]
 [  0.         0.         0.        37.063917  40.953486  45.251236  50.        55.24711   61.044864  67.451046  74.529508]
 [  0.         0.         0.         0.        33.543761  37.063917  40.953486  45.251236  50.        55.24711   61.044864]
 [  0.         0.         0.         0.         0.        30.357933  33.543761  37.063917  40.953486  45.251236  50.      ]
 [  0.         0.         0.         0.         0.         0.        27.474679  30.357933  33.543761  37.063917  40.953486]
 [  0.         0.         0.         0.         0.         0.         0.        24.865264  27.474679  30.357933  33.543761]
 [  0.         0.         0.         0.         0.         0.         0.         0.        22.503678  24.865264  27.474679]
 [  0.         0.         0.         0.         0.         0.         0.         0.         0.        20.366385  22.503678]
 [  0.         0.         0.         0.         0.         0.         0.         0.         0.         0.        18.432081]]

Option Price Grid
[[ 7.692854 10.767188 14.752139 19.753524 25.819765 32.932705 41.031823 50.078698 60.12262  71.268699 83.633082]
 [ 0.        4.4531    6.584668  9.529017 13.457729 18.493938 24.665995 31.887631 40.022542 49.059275 59.092951]
 [ 0.        0.        2.188192  3.465315  5.382525  8.168657 12.054357 17.195436 23.559177 30.868208 38.992873]
 [ 0.        0.        0.        0.821019  1.416741  2.412833  4.040968  6.619501 10.516207 15.968455 22.529508]
 [ 0.        0.        0.        0.        0.179063  0.344276  0.66192   1.272637  2.446829  4.704385  9.044864]
 [ 0.        0.        0.        0.        0.        0.        0.        0.        0.        0.        0.      ]
 [ 0.        0.        0.        0.        0.        0.        0.        0.        0.        0.        0.      ]
 [ 0.        0.        0.        0.        0.        0.        0.        0.        0.        0.        0.      ]
 [ 0.        0.        0.        0.        0.        0.        0.        0.        0.        0.        0.      ]
 [ 0.        0.        0.        0.        0.        0.        0.        0.        0.        0.        0.      ]
 [ 0.        0.        0.        0.        0.        0.        0.        0.        0.        0.        0.      ]]
(Press [Enter] to continue...)
```

## Functions and Classes

### Helpers

**# Return PDF, CDF, and sample correlation**
```
def pdf(x):
    return 1 / math.sqrt(2*math.pi) * math.exp(-0.5 * (x ** 2))

def cdf(x):
    return 0.5 * (1 + math.erf(x/math.sqrt(2)))

def sample_correlation(x, y):
    ...
```

**# Implements Closed Form European Option** (takes spot, strike, t (time elapsed), T (expiry), iv (volatility), rate (risk-free rate), cost (repo rate), option_type (call/put) as arguments.

```
class BlackScholesOption:
    def __init__(self, spot, strike, t, T, iv, rate, cost = 0, option_type = 'call'):
```

**# Calculates its vega** (helper function to the newton method iv estimation)
```
    def __vega(self):
    def vega(self):
```

**# Calculates its price** (price is a catch-all function utilizing self.option_type to get the correct price.
```
    def __call(self):
    def __put(self):
    def price(self):
```

**# Helper function to print the option**
```
    def __str__(self):
```

**# Extends BlackScholesOption() class and implements Risk-neutral Binomial Tree** (takes spot, strike, T (time to maturity), iv (volatility), rate (risk-free rate), N (steps of tree), option_type (call/put), option_class (american/european) as arguments.

```
class BinomialPricer(BlackScholesOption):
    def __init__(self, spot, strike, T, iv, rate, N, option_type = 'call', option_class = 'american'):
        super().__init__(spot, strike, 0, T, iv, rate, option_type = option_type)
        ...
```

**# Helper function to print the option**
```
    def __str__(self):
        return "[S = {0:6f}, K = {1}, iv = {2}, N = {3}, u = {4:6f}, d = {5:6f}, p = {6:6f}, x = 
{7:6f}]".format(self.spot, self.strike, self.iv, self.N, self.u, self.d, self.p, self.x)
```

**# Create the binomial tree and calculate option price**, can applied on both European and American option.
# In the function, spot_grid is populated by a loop. Moving across column (*c*) means a time step, moving horizontally means
# an up-step, while moving down means a down-step. The resulting grid is upper triangular. Some logic (such as internal
# symmetry of the spot grid is used to save compute.

# After spot grid is populated, the option price would be calculated from the back of the grid (c = steps), moving back in
# time. If the option is **American**, another step is added to compare the best price.

# A verbose output option is added to print the grid to console when steps <= 10 to help visualization as seen in demo.

```
    def price(self, verbose = False):
        spot_grid = np.zeros((self.N + 1, self.N + 1))
        for r in range(self.N + 1):
            for c in range(self.N + 1):
                ...

        price_grid = np.zeros((self.N + 1, self.N + 1))
        for c in reversed(range(self.N + 1)):
            for r in range(self.N + 1):
```

```
            ...

        if verbose:
            np.set_printoptions(precision = 6, suppress = True)
            print("\nSpot Price Grid")
            print(spot_grid)

            print("\nOption Price Grid")
            print(price_grid)

        return price_grid[0, 0]
```

    # A **recursion** version which almost always do **NOT** work when steps > 20.

```
    def price_recursion(self):
```

# **Extends BlackScholesOption() class and implements Asian Option** (takes spot, strike, T (time to maturity), iv (volatility), rate (risk-free rate), steps (n observation lookback), option_type (call/put), option_class (arithmetic/geometric) as arguments.

```
class AsianOption(BlackScholesOption):
    def __init__(self, spot, strike, T, iv, rate, steps, option_type = 'call', option_class = 
'geometric'):
```

    # Closed-form pricing function

```
    def price(self):
```

    # Standard Monte Carlo pricing
    # Create multiple paths simulating asset price movement (number determined by input `iterations`), and calculate the
    # average arithmetic/geometric mean price of that simulation to give the asset price and option price.

    # If option is geometric payoff, outputs the closed form price and standard MC price.
    # If option is arithmetic payoff, prompt user to select control. In case of control variate, calculate the best fit using the
    # simulated geometric payoff and closed form payoff, and output the confidence interval of the option price.

```
    def price_monte_carlo(self, iterations, control = True):
        drift = math.exp( (self.rate - 0.5 * self.iv ** 2) * self.T / self.steps )
        ...

        meanGP  = np.mean(listGP)
        meanAP  = np.mean(listAP)
        sdAP    = np.std (listAP, ddof = 1)
        pxClose = self.price()

        if self.option_class == 'geometric':
            ...

        if self.option_class == 'arithmetic':
            if control:
                ...
            else:
                ...
```

# **Partially extends BlackScholesOption() class as a class member and implements Basket Option** (takes S1, S2 (asset resp. spot price), strike, T (time to maturity), v1, v2 (asset resp. volatility), rate (risk-free rate), corr (asset correlation), option_type (call/put), option_class (arithmetic/geometric) as arguments.

```
class BasketOption():
    def __init__(self, S1, S2, v1, v2, rate, T, strike, corr, option_type = 'call', option_class = 
'geometric'):
```

    # Closed-form pricing function

```
    def price(self):
```

    # Standard Monte Carlo pricing

```
    # Create 2 correlated normal random variable (r_S1, r_S2) using 2 i.i.d. normal r.v. r_S1, r_S3 and simplified Cholesky
    # decomposition equation r_S2 = corr * r_S1 + sqrt (1 - corr ^2) * r_S3
    # Simulate individual asset price at maturity (number determined by input iterations), and calculate the
    # average arithmetic/geometric mean price of that simulation to give the asset prices and option price.

    # If option is geometric payoff, outputs the closed form price and standard MC price.
    # If option is arithmetic payoff, prompt user to select control. In case of control variate, calculate the best fit using the
    # simulated geometric payoff and closed form payoff, and output the confidence interval of the option price.

    def price_monte_carlo(self, iterations, control = True):
        drift1 = math.exp( ( self.rate - 0.5 * self.a1.iv ** 2 ) * self.T )
        drift2 = math.exp( ( self.rate - 0.5 * self.a2.iv ** 2 ) * self.T )

        np.random.seed(888)
        arrA1 = np.random.normal(0, 1, iterations)
        np.random.seed(444)
        arrA3 = np.random.normal(0, 1, iterations)
        arrA2 = self.corr * arrA1 + np.sqrt(1 - self.corr ** 2) * arrA3
        #print("corr(Z1, Z2) = ",sample_correlation(arrA1, arrA2)) = 0.5

        arrR1 = np.exp(self.a1.iv * math.sqrt( self.T ) * arrA1) * drift1
        arrR2 = np.exp(self.a2.iv * math.sqrt( self.T ) * arrA2) * drift2
        arrPA = (self.a1.spot * arrR1 + self.a2.spot * arrR2) / 2
        arrPG = (self.a1.spot * arrR1 * self.a2.spot * arrR2) ** 0.5

        ...
        pxClose = self.price()

        if self.option_class == 'geometric':
            ...

        if self.option_class == 'arithmetic':
            if control:
                ...

            else:
                ...

# Implements the newton method to estimate option IV from option premium in market, takes spot price (spot), strike (strike), t
(time elapsed), T (expiry time), r (risk-free rate), q (repo rate), mkt_price (option premium), option_type (call/put), and optionally
takes a initial guess.
# Terminates on reasonable precision reached or iterative limit reached.
# Could explode upon vega approaches 0 (if very outstrike or mispriced).

def iv_newton_method(spot, strike, t, T, r, q, mkt_price, option_type, guess = None):
    if guess is None:
        guess = math.sqrt(2 * abs( (math.log(spot/strike) + (r - q)*(T - t))/(T - t)))

    prev_guess = 0
    i = 0
    target_price = mkt_price

    while i < 100 and abs(guess - prev_guess) > 1e-8 :
        target = BlackScholesOption(spot,strike,t,T,guess,r,q)
        target_vega = target.vega()
        prev_guess = guess
        guess = guess - (target.price() - target_price)/target_vega
        print("next guess: ","{:.6f}".format(guess),"; last guess: ","{:.6f}".format(prev_guess),";
vega: ","{:.6f}".format(target_vega))
        i += 1

    print("implied volatility: {0:6f}".format(guess))
    return guess
```

## Test Cases (only geometric test cases results listed here; arithmetic test cases please find in the script)

### Asian

```
Test Case: Vol = 0.3, K = 100, n obs = 50, option type = put
[Geometric Asian from Standard MC: 8.557693, Closed-form price: 8.482705]

Test Case: Vol = 0.3, K = 100, n obs = 100, option type = put (put price drop after n obs increased)
[Geometric Asian from Standard MC: 8.518146, Closed-form price: 8.431080]

Test Case: Vol = 0.4, K = 100, n obs = 50, option type = put (put price higher after iv increased)
[Geometric Asian from Standard MC: 12.660593, Closed-form price: 12.558769]

Test Case: Vol = 0.3, K = 100, n obs = 50, option type = call
[Geometric Asian from Standard MC: 13.162008, Closed-form price: 13.259126]

Test Case: Vol = 0.3, K = 100, n obs = 100, option type = call (call price drop after n obs increased)
[Geometric Asian from Standard MC: 13.004248, Closed-form price: 13.138779]

Test Case: Vol = 0.4, K = 100, n obs = 50, option type = call (call price higher after iv increased)
[Geometric Asian from Standard MC: 15.641245, Closed-form price: 15.759820]
```

**Conclusion**: Asian options behaves largely like vanilla, but both put and call price tends to **decrease** when n obs **increases**

### Basket

```
Test Case: S0_1 = 100, S0_2 = 100, K = 100, v_1 = 0.3, v_2 = 0.3, corr = 0.5, option type = put
[Geometric Basket from Standard MC: 11.463490, Closed-form price: 11.491573]
(put option price higher if correlation higher)
Test Case: S0_1 = 100, S0_2 = 100, K = 100, v_1 = 0.3, v_2 = 0.3, corr = 0.9, option type = put
[Geometric Basket from Standard MC: 12.589602, Closed-form price: 12.622350]
(put option price lower if either asset iv lower)
Test Case: S0_1 = 100, S0_2 = 100, K = 100, v_1 = 0.1, v_2 = 0.3, corr = 0.5, option type = put
[Geometric Basket from Standard MC: 6.580378, Closed-form price: 6.586381]
(put option price lower if either asset strike lower)
Test Case: S0_1 = 100, S0_2 = 100, K = 80, v_1 = 0.3, v_2 = 0.3, corr = 0.5, option type = put
[Geometric Basket from Standard MC: 4.703026, Closed-form price: 4.711577]
(put option price higher if either asset strike higher)
Test Case: S0_1 = 100, S0_2 = 100, K = 120, v_1 = 0.3, v_2 = 0.3, corr = 0.5, option type = put
[Geometric Basket from Standard MC: 21.249924, Closed-form price: 21.289105]
(put option price higher if both asset iv higher)
Test Case: S0_1 = 100, S0_2 = 100, K = 100, v_1 = 0.5, v_2 = 0.5, corr = 0.5, option type = put
[Geometric Basket from Standard MC: 23.425374, Closed-form price: 23.469148]

Test Case: S0_1 = 100, S0_2 = 100, K = 100, v_1 = 0.3, v_2 = 0.3, corr = 0.5, option type = call
[Geometric Basket from Standard MC: 22.244795, Closed-form price: 22.102093]
(call option price higher if correlation higher)
Test Case: S0_1 = 100, S0_2 = 100, K = 100, v_1 = 0.3, v_2 = 0.3, corr = 0.9, option type = call
[Geometric Basket from Standard MC: 26.075112, Closed-form price: 25.878826]
(call option price lower if either asset iv lower)
Test Case: S0_1 = 100, S0_2 = 100, K = 100, v_1 = 0.1, v_2 = 0.3, corr = 0.5, option type = call
[Geometric Basket from Standard MC: 18.006384, Closed-form price: 17.924737]
(call option price higher if either asset strike lower)
Test Case: S0_1 = 100, S0_2 = 100, K = 80, v_1 = 0.3, v_2 = 0.3, corr = 0.5, option type = call
[Geometric Basket from Standard MC: 32.698491, Closed-form price: 32.536256]
(call option price lower if either asset strike higher)
Test Case: S0_1 = 100, S0_2 = 100, K = 120, v_1 = 0.3, v_2 = 0.3, corr = 0.5, option type = call
[Geometric Basket from Standard MC: 14.817070, Closed-form price: 14.685466]
(call option price higher if both asset iv higher)
Test Case: S0_1 = 100, S0_2 = 100, K = 100, v_1 = 0.5, v_2 = 0.5, corr = 0.5, option type = call
[Geometric Basket from Standard MC: 28.707434, Closed-form price: 28.449387]
```

**Conclusion**: Basket options behaves largely like vanilla, price would be impacted by either asset parameter moving, while correlation **increasing** tend to **increase** the option price for both put and calls.