Final Assignment – Programming for finance with VBA

Haoyang Wang | Jian Shen

31/10/2020

1 Overview

One important feature of this project is anyone could put any size of data and immediately get the result from this data by clicking one button.

All of inputs/outputs are stored in vba array because array takes less memory and is calculation inexpensive compared with range object, especially when doing Monte Carlo simulation

In this project, I used dictionary object a lot to store intermediate result. For example, I use dictionary to store 3 typical pattern interest rate paths and stock price simulation under these patterns.

Because VBA is not good at vectorized and matrix operation and if one wants to use array as basic unit, he/she has to count on "for loop" to handle calculation. For example, the calculation of Treynor ratio needs parameters like weights(vector), mu(vector), returns (matrix), which make the codes not easy to read because of for loop.

1.1 Modules

There are 5 modules(including one class module) in this VBA project(FinalAssignment.xlsm)

- **answer**: main module for presenting the answer for questions
- **fatory**: contains functions to set a new class with parameters passed in
- **helper_function**: most important module for the project. It contains many useful financial functions, for example, VaR, CVaR, Merton Rate simulation, stock price simulation with rates obtained from Merton model, select 3 typical rates path, payoff function for Eu, Am, digital, spread option, BSM, greeks, CRR binomial models for combination of call/put, digital, spread Eu/Am options(one function for all different options). You could see full list of functions from this module in Appendix- Financial Functions
- **utili_function**: format and print result, sorting items for dictionary objects(select 3 typical rates paths in Merton Model)

1.2 Class Modules

The main idea I use class module is that it's easy to test and debug and put all the relevant procedures in one place, which makes codes more manageable in later phase.

1.2.1 cls_assets

This class module mainly works for Question 1.

Initiate a cls_assets with following function

```
CreateClsAsset(assetPrice As Range, tickerName As Range, Dates As Range)
```

- assetPrice: 2-demension excel range containing assets price, e.x. Range("B2", "D11")
- tickerName: 1-demension excel range
- Dates: 1-demension excel range

Property

```
Property Get expectedRet()
Property Get volatility()
Property Get VaR_(confidence As Double)
Property Get CVaR_(confidence As Double)
Property Get CovMatrix()
Property Get CorrMatrix()
Property Get numAssets(): number of total assets
Property Get numRows(): number of records
Property Get returnsDaily()
Property Get SharpeRatio()
Property Get SortinoRatio(rf As Double, weights As Range)
Property Get TreynorRatio(rf As Double, weights As Range)
...
```

Method

```
Initiate(assetPrice As Range, tickerName As Range, Dates As Range)
```

The method `Initiate()` helps to get the basic statistics like mu and standard deviation at the time when class is built

1.2.2 cls_options

This class module mainly works for Question 2 & 3.

Initiate a cls_assets with following function

```
CreateClsOption(K_ As Double, S0_ As Double, q_ As Double, T_ As Double)
```

Property

```
Property Get optionPrice_Merton()
```

Method

`Initiate(K_ As Double, S0_ As Double, q_ As Double, T_ As Double)`

`RateSim_3TypicalPattern((startValue As Double, mu As Double, sigma As Double, Nsteps As Integer, nSim As Integer, seed As Integer))`

`StockPriceSim_Merton(startValue As Double, mu As Double, sigma As Double, Nsteps As Integer, nSim As Integer, S0 As Double, stockSigma As Double, StocknSim As Integer, seed As Integer)`

`Greeks(S As Double, K As Double, vol As Double, rf As Double, T As Double, q As Double, numPoints As Integer, Optional precision As Double = 100)`

`CRR_Tree(S0 As Double, K As Double, vol As Double, rf As Double, T As Double, Nsteps As Integer, iscall As Boolean, isAmerican As Boolean, printAddress As String, Optional isdigital As Boolean = False, Optional isSpread As Boolean = False, Optional S2 As Double)`

The CRR_Tree() Method helps to play with any combination of option type by specifying the arugments: iscall, isAmerican, isdigital, isSpread.

2 Question 1 – Preliminary Analysis – 4 points

2.1 Introdution

- Prices data is in tab "price_data", but one can replace it with his/her own data (any size) and modify the input parameters in `factory.CreateClsAsset` from macro question1() in module "answer".
- The assets selected could be divided into the following sector:
  - technology: AAPL, AMD, ADI, AMZN, MSFT, GOOGL
  - manufacture: TSLA, BA
  - financial: JPM
  - entertainment: NFLX
  - index: IXIC, GSPC
  - bonds: DTB3
  - healthcare: ABBV, AEZS, A.
  - basic materials: APD, AA, CF.
- Daily simple returns is calculated and stored in a separate tab "ret_data".

2.2 Descriptive Statistics:

Table 1: Descriptive statistics (only show 10 assets)

	AAPL	AMD	ADI	ABBV	AEZS	MSFT	GOOGL	X.IXIC	X.GSPC	DTB3
Ret_d	0.0036	0.0043	0.0007	0.0008	0.0002	0.0027	0.0014	0.0018	0.0008	0.0000
mu_ann	0.9056	1.0727	0.1858	0.2120	0.0486	0.6726	0.3633	0.4642	0.1942	0.0033
vol_d	0.0312	0.0427	0.0368	0.0242	0.0913	0.0311	0.0265	0.0253	0.0254	0.0000
vol_ann	0.4954	0.6785	0.5848	0.3834	1.4499	0.4936	0.4210	0.4018	0.4027	0.0003
var_d	0.0010	0.0018	0.0014	0.0006	0.0083	0.0010	0.0007	0.0006	0.0006	0.0000
var_ann	0.2455	0.4603	0.3420	0.1470	2.1022	0.2436	0.1773	0.1614	0.1621	0.0000

2.3 Covariance Matrix & Correlation Matrix

Table 2: Covaraince matrix (only show 10 assets)

	AAPL	AMD	ADI	ABBV	AEZS	MSFT	GOOGL	X.IXIC	X.GSPC	DTB3
AAPL	0.2455	0.2223	0.2022	0.1243	0.1572	0.2124	0.1688	0.1799	0.1738	0
AMD	0.2223	0.4603	0.2449	0.1315	0.2373	0.2358	0.1876	0.2004	0.1853	0
ADI	0.2022	0.2449	0.3420	0.1219	0.2448	0.2189	0.1838	0.1980	0.1951	0
ABBV	0.1243	0.1315	0.1219	0.1470	0.1153	0.1304	0.1097	0.1119	0.1159	0
AEZS	0.1572	0.2373	0.2448	0.1153	2.1022	0.1295	0.1591	0.1575	0.1531	0
A	0.1535	0.1614	0.1659	0.1058	0.1277	0.1558	0.1304	0.1383	0.1431	0
APD	0.1769	0.1879	0.1974	0.1196	0.1141	0.1804	0.1500	0.1597	0.1690	0
AA	0.2318	0.2594	0.3426	0.1364	0.3793	0.2207	0.2087	0.2272	0.2459	0
CF	0.1892	0.1956	0.2584	0.1377	0.1927	0.1865	0.1683	0.1839	0.2010	0
IBM	0.1616	0.1751	0.1894	0.1134	0.1800	0.1645	0.1421	0.1527	0.1637	0

Table 3: Correlation matrix (only show 10 assets)

	AAPL	AMD	ADI	ABBV	AEZS	MSFT	GOOGL	X.IXIC	X.GSPC	DTB3
AAPL	1.0000	0.6613	0.6977	0.6543	0.2188	0.8687	0.8092	0.9039	0.8711	-0.0837
AMD	0.6613	1.0000	0.6174	0.5055	0.2412	0.7040	0.6568	0.7353	0.6784	-0.0692
ADI	0.6977	0.6174	1.0000	0.5437	0.2888	0.7583	0.7463	0.8427	0.8285	-0.0461
ABBV	0.6543	0.5055	0.5437	1.0000	0.2075	0.6889	0.6797	0.7264	0.7509	-0.0231
AEZS	0.2188	0.2412	0.2888	0.2075	1.0000	0.1809	0.2606	0.2703	0.2622	-0.0055
A	0.7412	0.5689	0.6785	0.6598	0.2106	0.7551	0.7407	0.8235	0.8498	-0.0789
APD	0.7301	0.5661	0.6902	0.6377	0.1608	0.7470	0.7282	0.8127	0.8579	-0.0434
AA	0.5086	0.4156	0.6367	0.3866	0.2843	0.4859	0.5386	0.6146	0.6637	-0.1248
CF	0.5875	0.4435	0.6798	0.5527	0.2045	0.5814	0.6149	0.7043	0.7681	-0.0779
IBM	0.7091	0.5613	0.7042	0.6430	0.2699	0.7246	0.7340	0.8262	0.8840	-0.0228

2.4 VaR & CVaR

Table 4: VaR and CVaR (only show 10 assets)

	AAPL	AMD	ADI	ABBV	AEZS	MSFT	GOOGL	X.IXIC	X.GSPC	DTB3
VaR95	0.0551	0.0748	0.0615	0.0407	0.1509	0.0540	0.0452	0.0436	0.0426	0e+00
VaR98	0.0679	0.0923	0.0766	0.0506	0.1883	0.0667	0.0561	0.0540	0.0530	0e+00
CVaR95	0.0682	0.0927	0.0770	0.0508	0.1892	0.0670	0.0563	0.0542	0.0533	0e+00
CVaR95	0.0794	0.1080	0.0902	0.0595	0.2220	0.0782	0.0658	0.0633	0.0624	1e-04

Note: one also could set any confidence level in `Get VaR_(confidence As Double)`, `Get CVaR_(confidence As Double)` property from `cls_assets`

2.5 Sharpe, Treynor and Sortino Ratios

For simplicity, I demonstrated the equally weighted portfolio (but you could enter any weights for calculation of these ratios in `Property Get SharpeRatio()`, `Property Get SortinoRatio(rf As Double, weights As Range)`, `Property Get TreynorRatio(rf As Double, weights As Range)`, the default for weights is $1/N$)

Table 5: Sharpe, Treynor and Sortino Ratios

	Ratios
Sharpe	2.3703
Treynor	0.4264
Sortino	0.9943

Note:

- It's not easy in VBA to write function which is adapted to dynamic changing range size, for example one needs weights and covariance matrix to calculate Sharpe ratio but the length of weights and covariance matrix differs each time.
- Please note that we couldn't calculation Treynor ratio with new data. (because one needs to pass in market daily return for the calculation. The current Treynor ratio is using ^GSPC in column 19 to represent market performance in the calculation)

2.6 Summary and Comment

I used 3 month treasury bills("DTB3") as risk free rate approximation, and sp500 index("X.GSPC") to represent market performance.

From descriptive statistics, see Table 1, AAPL and AMD performs very well with 90.56% and 107% annual return respectively while the index SP500 and Nasdq are 46.42% and 19.42% respectively.

From correlation matrix (Table 3) and the heatmap (Figure 1)chart below, the return series shows a very large linear correlation except for the 3month treasury bills. It's mainly because we selected companies which perform very well in Covid period.

	AAPL	AMD	ADI	ABBV	AEZS	A	APD	AA	CF	IBM	AMZN	NFLX	JPM	BA	TSLA	MSFT	GOOGL	X.IXIC	X.GSPC	DTB3
AAPL	1	0.661328	0.697719	0.654257	0.218813	0.741228	0.730092	0.508554	0.587481	0.709095	0.667322	0.563145	0.652945	0.522961	0.467054	0.868681	0.809188	0.903875	0.871055	-0.08366
AMD	0.661328	1	0.617368	0.505499	0.241223	0.568907	0.566138	0.415608	0.443461	0.561254	0.634035	0.533471	0.453068	0.338703	0.430837	0.704002	0.656814	0.73533	0.678376	-0.06921
ADI	0.697719	0.617368	1	0.543673	0.288758	0.678454	0.690203	0.636681	0.679752	0.704151	0.55068	0.449881	0.706911	0.536334	0.48678	0.75828	0.746275	0.842722	0.828539	-0.04608
ABBV	0.654257	0.505499	0.543673	1	0.207486	0.65975	0.637706	0.38664	0.552666	0.643038	0.416417	0.378016	0.60074	0.444106	0.353079	0.688933	0.679748	0.726393	0.75092	-0.02305
AEZS	0.218813	0.241223	0.288758	0.207486	1	0.210585	0.160833	0.284318	0.204507	0.269863	0.166589	0.191294	0.190926	0.285334	0.229728	0.180896	0.260645	0.270307	0.262214	-0.00546
A	0.741228	0.568907	0.678454	0.65975	0.210585	1	0.810123	0.487195	0.594212	0.75665	0.517531	0.400498	0.711881	0.514309	0.31734	0.755131	0.740655	0.823518	0.849763	-0.07895
APD	0.730092	0.566138	0.690203	0.637706	0.160833	0.810123	1	0.534931	0.622744	0.781918	0.501675	0.394046	0.74369	0.475432	0.348368	0.747046	0.728203	0.81265	0.857895	-0.0434
AA	0.508554	0.415608	0.636681	0.38664	0.284318	0.487195	0.534931	1	0.712102	0.659528	0.305053	0.253211	0.686553	0.640299	0.42773	0.485856	0.538615	0.614551	0.663704	-0.12482
CF	0.587481	0.443461	0.679752	0.552666	0.204507	0.594212	0.622744	0.712102	1	0.712308	0.323956	0.280403	0.789797	0.690232	0.400258	0.581368	0.614945	0.70432	0.768077	-0.07786
IBM	0.709095	0.561254	0.704151	0.643038	0.269863	0.75665	0.781918	0.659528	0.712308	1	0.537267	0.382797	0.784876	0.661946	0.361517	0.724589	0.733958	0.82618	0.883988	-0.02281
AMZN	0.667322	0.634035	0.55068	0.416417	0.166589	0.517531	0.501675	0.305053	0.323956	0.537267	1	0.696221	0.363163	0.271153	0.478963	0.718931	0.6944	0.734239	0.62008	-0.08067
NFLX	0.563145	0.533471	0.449881	0.378016	0.191294	0.400498	0.394046	0.253211	0.280403	0.382797	0.696221	1	0.303315	0.247763	0.440723	0.62601	0.604616	0.633195	0.518928	-0.00668
JPM	0.652945	0.453068	0.706911	0.60074	0.190926	0.711881	0.74369	0.686553	0.789797	0.784876	0.363163	0.303315	1	0.707592	0.362567	0.673249	0.670112	0.769068	0.860851	-0.05413
BA	0.522961	0.338703	0.536334	0.444106	0.285334	0.514309	0.475432	0.640299	0.690232	0.661946	0.271153	0.247763	0.707592	1	0.362447	0.502482	0.545221	0.6173	0.689273	-0.04843
TSLA	0.467054	0.430837	0.48678	0.353079	0.229728	0.31734	0.348368	0.42773	0.400258	0.361517	0.478963	0.440723	0.362567	0.362447	1	0.530409	0.516184	0.569813	0.496653	0.021818
MSFT	0.868681	0.704002	0.75828	0.688933	0.180896	0.755131	0.747046	0.485856	0.581368	0.724589	0.718931	0.62601	0.673249	0.502482	0.530409	1	0.878123	0.942999	0.893535	-0.03682
GOOGL	0.809188	0.656814	0.746275	0.679748	0.260645	0.740655	0.728203	0.538615	0.614945	0.733958	0.6944	0.604616	0.670112	0.545221	0.516184	0.878123	1	0.91727	0.878533	-0.04911
X.IXIC	0.903875	0.73533	0.842722	0.726393	0.270307	0.823518	0.81265	0.614551	0.70432	0.82618	0.734239	0.633195	0.769068	0.6173	0.569813	0.942999	0.91727	1	0.971559	-0.07368
X.GSPC	0.871055	0.678376	0.828539	0.75092	0.262214	0.849763	0.857895	0.663704	0.768077	0.883988	0.62008	0.518928	0.860851	0.689273	0.496653	0.893535	0.878533	0.971559	1	-0.06938
DTB3	-0.08366	-0.06921	-0.04608	-0.02305	-0.00546	-0.07895	-0.0434	-0.12482	-0.07786	-0.02281	-0.08067	-0.00668	-0.05413	-0.04843	0.021818	-0.03682	-0.04911	-0.07368	-0.06938	1

Figure 1: Correlation matrix – heat map

The calculation method for VaR and CVaR is based on assumption that distribution of return follows normal distribution.

$$VaR_{1-\alpha} = \mu + \sigma z_{1-\alpha}$$

$$ES_{\alpha} = \mu + \sigma \frac{\phi(\Phi^{-1}(\alpha))}{\alpha}$$

where $\phi(x)$ is the standard normal p.d.f.
But one could also use historical data to get VaR and CVaR. I also code this method in VBA but comment them. Table 4 shows that "AEZS" has largest VaR and CVaR, meaning that, from the Value at Risk as risk measure, "AEZS" is most risky asset while "DTB3" is less risky and thus could being representative of risk free rate.

Table 6: Ratios compared with market performance

	1/N portfolio	SP500
Sharpe	2.3703	0.4740
Treynor	0.4264	0.1909
Sortino	0.9943	0.8953

As shown in Table 6 (ratios in SP500 could be found in tab "ans_1_appendix"), these indicators show that naive portfolio by far outperforms benchmark(SP500). The beta of naive portfolio is 1.0096 very close to market beta 1, but Treynor ratio is largely bigger than benchmark, which means the 1/N strategy earned a large excess return while taking the similar market risk(systematic risk). However, Sortino ratios (0.9943 v.s. 0.8953) are quite similar, which means 1/N stratey(in our case, not for general case) faces much more downside risk than benchmark. The Sharpe ratio is far way larger than benchmark, meaning portfolio is much more advantageous for taking per unit of risk.

The reason why 1/N strategy in our case outperforms could be

- The assets selected in this period mainly contains very large technology, entertainment, and health care companies. It's not strange that these assets would perform very well in this Covid-19 period.

3 Question 2 – Simulation and Pricing – 8 points

3.1 Introduction

- All the settings(arguments) could be modified in the left area of tab "ans_2"
- Please set random seed in cell "B5" as "2020" to repeat the simulation result
- Because of efficiency of using array for calculation and outputting the result, one could have a large simulation number in setting.
- Global arguments/parameters:
  - Strike price K = 100
  - Stock price initial S0 = 100
  - q = 0, you could change it in Option Greeks analysis part
  - Time range T = 5
- Arguments for interest rate simulation:
  - initial rate: 0.08
  - mu for interest rate: 0.003
  - sigma for interest rate: 0.02
  - Nsteps: 20, which is the same for stock price simulation. So one do not need to set it in "Stock Price Simulation" part
  - number of simulation for rate: 100
- Arguments for stock price simulation:
  - S0: 100
  - sigma: 0.23
  - number of simulation: 1000

3.2 Select 3 patterns of simulated r

I follow the advice from "final assignment" video: choosing best scenario, worst scenario and flat scenario. The criteria of "best" or "worst" depends on the gap between starting rate and terminal rate. i.e. if the terminal rate is greatest among all the path, then this path is selected as best scenario (this sorting process is done in dictionary object, thanks to PAUL KELLY).

One could see the numerical result of 3 typical pattern of interest rates evolution in Table 7 and notice that in worst case, interest rate goes below than zero, which should not be the case in reality(Also shown in Figure 2). One might replace Merton model with other interest rate model, like Vasicek model which has mean reverting property and could reduce probability that rate would be negative.

It's quite obvious from Figure 2 that the 3 selected paths meet our definition of what is best, worst and flat scenario.

Table 7: 3 patterns of simulated rate

Date	Best case	Worst case	Flat case
2020-09-01	0.0800	0.0800	0.0800
2020-12-01	0.0952	0.0704	0.0709
2021-03-01	0.0999	0.0707	0.0898
2021-06-01	0.1088	0.0744	0.0866
2021-09-01	0.1193	0.0743	0.0865
2021-12-01	0.1472	0.0599	0.0820
2022-03-01	0.1512	0.0450	0.0805
2022-06-01	0.1548	0.0371	0.0860
2022-09-01	0.1538	0.0268	0.0916
2022-12-01	0.1511	0.0202	0.0799
2023-03-01	0.1494	0.0175	0.0882
2023-06-01	0.1686	0.0051	0.0881
2023-09-01	0.1726	-0.0029	0.1037
2023-12-01	0.1655	-0.0220	0.1150
2024-03-01	0.1719	-0.0253	0.1101
2024-06-01	0.1766	-0.0286	0.1226
2024-09-01	0.1885	-0.0382	0.1136
2024-12-01	0.1992	-0.0446	0.0874
2025-03-01	0.2258	-0.0394	0.0927
2025-06-01	0.2274	-0.0345	0.0923
2025-09-01	0.2251	-0.0358	0.0903

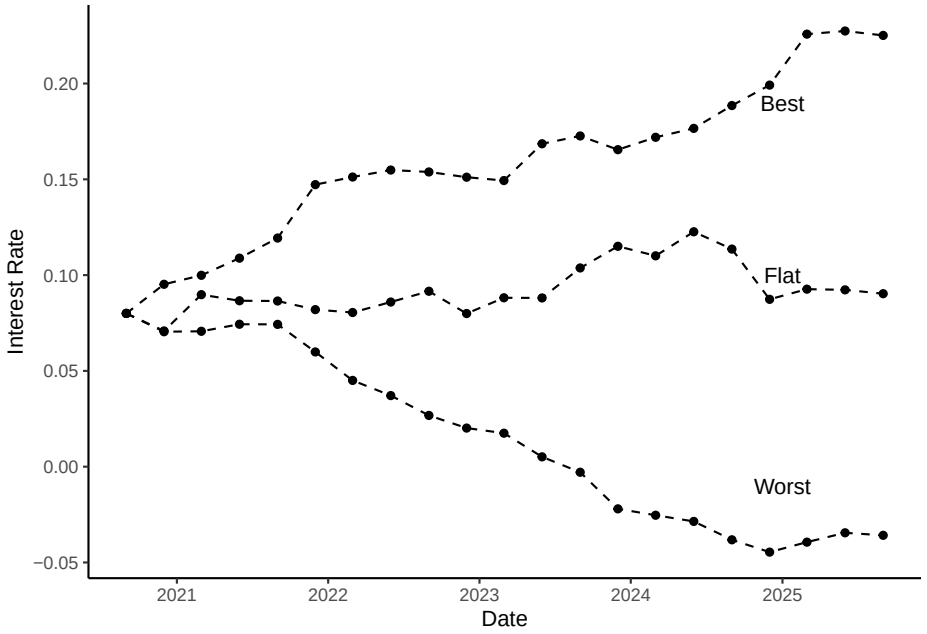

Figure 2: Interest rate evoltuion with 3 patters: best, worst and flat

3.3 GBM for stock price simulation

The simulation results of stock price with 3 different rate patterns are stored in tab "bestScenario", "worstScenario", "flatScenario" respectively. The main procedure for simulation is described as follows:

1. from n step to n + 1 step, use r_n as new risk free rate to recalculate $v_n dt = (r_n - 0.5\sigma^2)dt$
2. calculate new stock price: $S_{n+1} = S_n exp(v_n dt + \sigma\sqrt{dt}\epsilon)$

The Figure 3 shows the average stock price evolution with 3 rate patters(average stock price at each step among all the paths). It's understandable that with best scenario(rates increase most), the average stock price under this rate path also obtains highest price (because $v_n dt$ is much bigger).

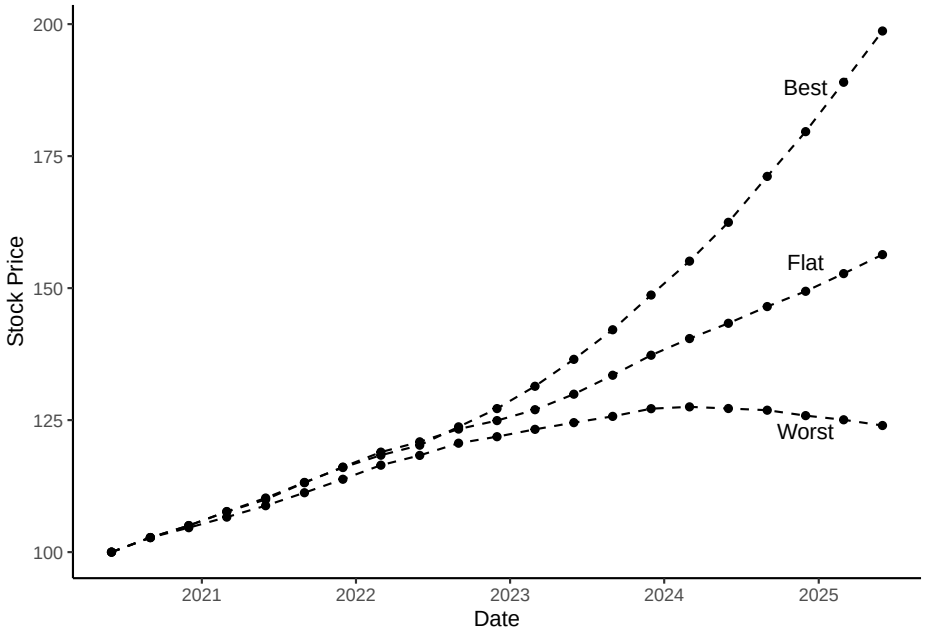

Figure 3: Stock average price evoltuion with 3 patters: best, worst and flat

3.4 Option price using stock price simulation

Sanity check of result:

Compare the result with BSM formula when interest rate is constant, that is, set the mu and sigma in rate simulation to 0. Table 8 shows that 5 years option prices is very close with that given by BSM formula.

Table 8: Option price when rate is constant

	Call Price	Put Price
Best Scenario	39.5696	4.8486
Worst Scenario	39.5696	4.8486
Flat Scenario	39.5696	4.8486
BSM	38.1304	5.1624

Option price under 3 rates path:

Table 9: Option price under 3 rate paths

	Call Price	Put Price
Best Scenario	57.7661	0.9055
Worst Scenario	24.4050	15.1973
Flat Scenario	42.9107	3.8620
BSM	38.1304	5.1624

Table 9 shows the call/put of Europe option price under 3 rate paths differs a lot, with price under flat path being closest to BSM result. Also, the price in best scenario obtains highest price because as one can see from Figure 3, the average sock price in best scenario increases most, which make the option deep in the

money in the future, while put option price in the same scenario behaves oppositely. Similar analysis can apply to worst and flat case.

Note: The calculation of option price is a little different because the discount factor differs in each step as shown in the following equation:

$$V_{option} = \frac{1}{N}\sum_{i=1}^{N} exp(-\sum_{i=j}^{steps} r_j dt)\ payoff(S^{(i)}{}_T, K)$$

where r_j is the simulated rate in step j.

3.5 Option Greeks

3.5.1 Arguments for option greeks:

- S0: 100
- K: 100
- sigma: 0.4
- rf: 0.08
- T: 1
- q: 0.06 (for choosing a large q will help to identify the change in option price with change in dividend yield. Also, do not choose q equal to rf, which will make the price of put and call equal)
- *points generated: 20 (total number of points generated for Greeks analysis)
- *precision: 12 (The larger the precision is, the smaller the change of the dependant variable)

3.5.2 The calculation method:

For the calculation method, I didn't use the closed form solution for greeks as it only applies to Plain Vanilla Option. I calculate the sensitivity using the discrete method as shown below:

$$Greeks = \frac{\partial V}{\partial X} \approx \frac{\Delta V}{\Delta X}$$

where V is value of option and X could be underlying asset price, risk free rate, dividend yield.

3.5.3 Results

Table 10: Option Greeks (only show 9 columns)

	V2	V3	V4	V5	V6	V15	V16	V17	V18
rf	0.0133	0.0200	0.0267	0.0333	0.0400	0.1000	0.1067	0.1133	0.1200
price	13.1384	13.3868	13.6377	13.8911	14.1469	16.5546	16.8333	17.1140	17.3968
Rho_C	36.8882	37.2650	37.6384	38.0081	38.3741	41.4767	41.7978	42.1136	42.4241
price	17.6374	17.2302	16.8298	16.4363	16.0494	12.8619	12.5393	12.2229	11.9124
Rho_P	-62.1170	-61.0823	-60.0555	-59.0366	-58.0258	-49.3093	-48.3850	-47.4700	-46.5642
sigma	0.0667	0.1000	0.1333	0.1667	0.2000	0.5000	0.5333	0.5667	0.6000
price	3.5226	4.7248	5.9442	7.1695	8.3968	19.3544	20.5527	21.7458	22.9333
Vega_C	34.0072	36.0669	36.5807	36.7599	36.8192	36.0953	35.9491	35.7928	35.6268
price	1.6578	2.8600	4.0794	5.3047	6.5320	17.4896	18.6879	19.8810	21.0685
Vega_p	34.0072	36.0669	36.5807	36.7599	36.8192	36.0953	35.9491	35.7928	35.6268
T	0.1667	0.2500	0.3333	0.4167	0.5000	1.2500	1.3333	1.4167	1.5000
price	6.5981	8.0755	9.3102	10.3868	11.3492	17.3828	17.8796	18.3535	18.8062

	V2	V3	V4	V5	V6	V15	V16	V17	V18
Theta_C	-23.2499	-17.7293	-14.8157	-12.9194	-11.5487	-6.2634	-5.9626	-5.6865	-5.4317
price	6.2686	7.5842	8.6589	9.5774	10.3836	15.0921	15.4505	15.7876	16.1051
Theta_P	-21.2846	-15.7870	-12.8961	-11.0224	-9.6739	-4.5809	-4.3006	-4.0447	-3.8099
q	0.0100	0.0150	0.0200	0.0250	0.0300	0.0750	0.0800	0.0850	0.0900
price	18.7388	18.4209	18.1068	17.7966	17.4903	14.9023	14.6332	14.3676	14.1056
Epsilon_C	-64.3647	-63.5847	-62.8087	-62.0368	-61.2691	-54.5558	-53.8325	-53.1139	-52.4001
price	12.0454	12.2213	12.3986	12.5773	12.7574	14.4396	14.6332	14.8280	15.0241
Epsilon_P	34.8882	35.1732	35.4566	35.7384	36.0185	38.4509	38.7103	38.9673	39.2219

3.5.4 Rho

Rho, ρ, measures sensitivity to the interest rate. As shown in Figure 4, rho is positive with call option and increases when risk rates increase. One also could notice that in our case, the absolute value rho of put option is larger than call option, meaning price of put option is more sensitive to the change of risk free rate. Besides, the slope in put option case is bigger than that in call option, meaning ρ of put option is more sensitive to the change of risk free rate in our case.

Question: what happens to the Option Price (both Call and Put) following different changes in the Rate? Explain

Answer:

As risk free rate increase, call option will also increase while the put option price decreases (see the charts in excel tab "ans2_graph").
Reason is simple, as one could see from Stochastic Differential Equation (SDE) of stock price Geometric Brownian Motion, the drift term would increase as risk free rate increase in neutral probability measure world, which leads to a higher stock price in the end with more possibility.

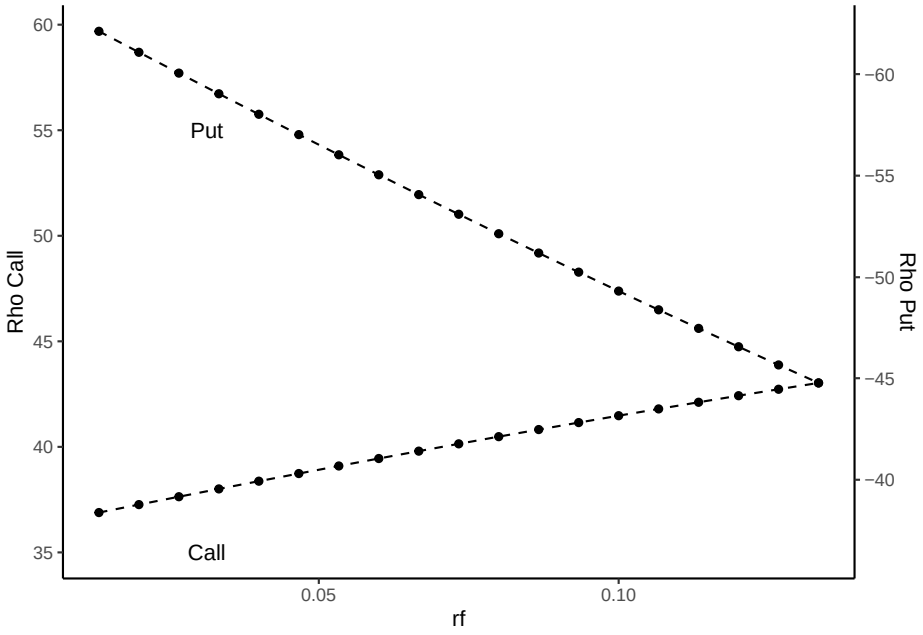

Figure 4: Option price sensitivity to the interest rate

3.5.5 Vega

Vega, ν, measures sensitivity to volatility. As shown in Figure 5, ν increase sharply at first and decrease slowly after reaching the highest point at volatility around 0.2 in our case. Vega of call option and put option is same, meaning the volatility has same impact on the call/put option value.

what happens to the Option Price (both Call and Put) following different changes in the Volatility? Explain;

Answer:

For both call and put option, the price will increase as volatility grows (see chart in the excel). And the change of increase is same because the vega is same for both option. One also could explain it in stock price GBM. Because volatility increases, the final price of stock price would be more volatile, underwriter should be rewarded for bearing more risk.

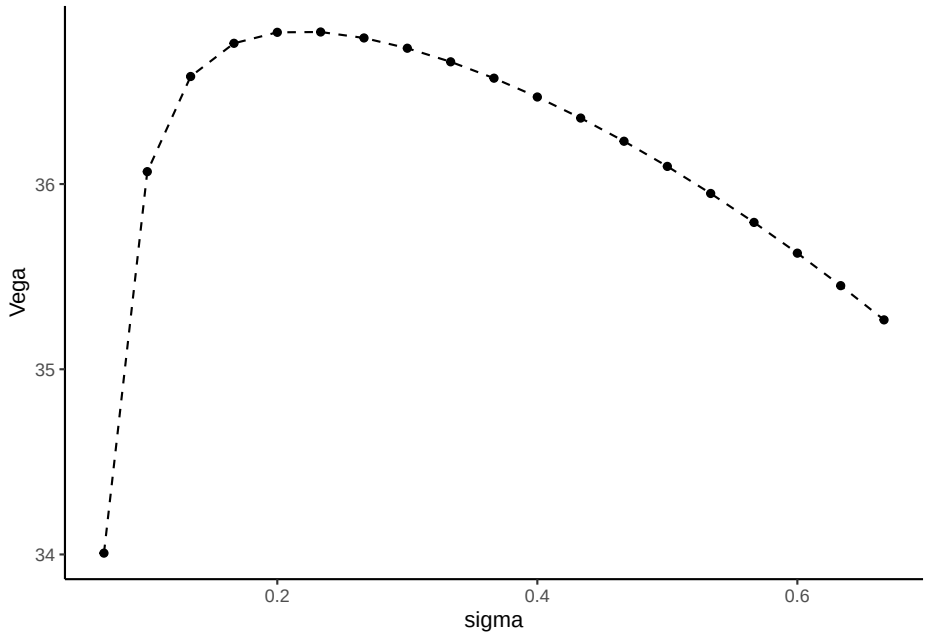

Figure 5: Option price sensitivity to the volatiliy

3.5.6 Theta

Theta, θ, measures the sensitivity of the value of the derivative to the passage of time: the "time decay.". As shown in Figure 6, both θ of put and call option behave similar pattern: increase with the increase in time to expiration. Theta is almost always negative for long calls and puts, which is also the case in our example.

Question: what happens to the Option Price (both Call and Put) following different changes in the Time to expiration?

Answer: As being closer to the maturity, the price decrease much more. (The time value declines or time decay accelerates as the expiration date gets closer because there's less time for an investor to earn a profit from the option). This is also proved in the theta graph (Figure 6) when near the maturity, theta is most negative(price decay most).

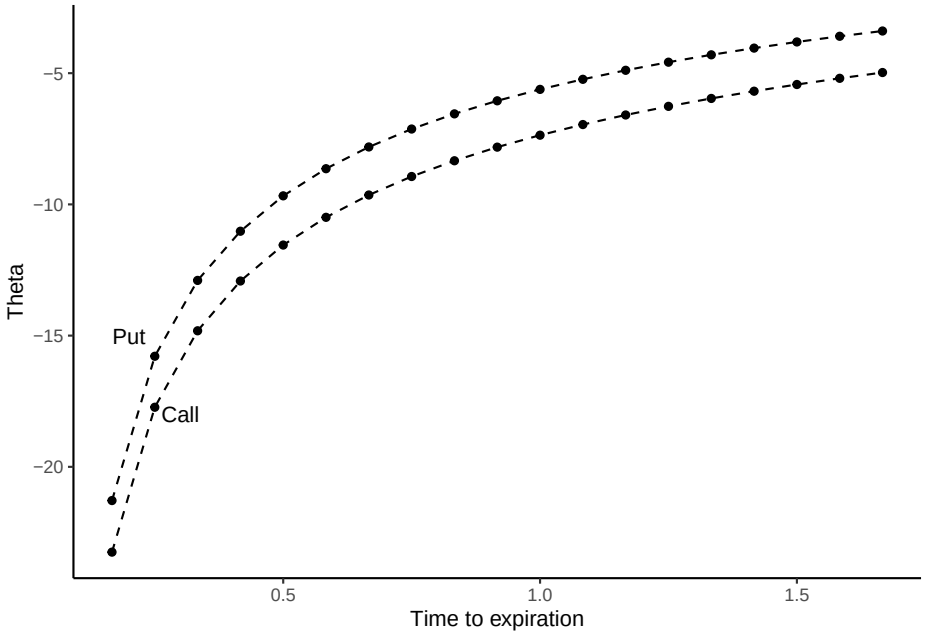

Figure 6: Option price sensitivity to the time to expiration

3.5.7 Epsilon

Epsilon, ϵ, is the percentage change in option value per percentage change in the underlying dividend yield, a measure of the dividend risk. The epsilon in our case is negative in call option while positive in put option. With the increase in dividend yield, the epsilon in call option increases while decreases in put option. But for per unit of change in dividend yield, the epsilon of call option is more sensitive as the slope of call in Figure 7 is more steep.

Question: what happens to the Option Price (both Call and Put) following different changes in the Dividend yield?

Answer:

Value of call option decline while put option increase when dividend yield grows. The effect of change in dividend yield on option price is opposite to that in risk free rate.

The reason can be explained very intuitively: a bigger dividend yield will lead the stock price decreasing. For example, when the dividend is allocated to the shareholder, the stock price will decrease. So the final stock price at maturity would be not bigger than the case with no dividend. Or one can see it in this replication(delta hedge) way: you try to hold delta stocks to hedge the short position of option. when you have the dividend, you could use the dividend to buy the new shares at each stage. So the initial investment to buy delta number of stocks would be smaller than the case with no dividend, which make the initial investment smaller.

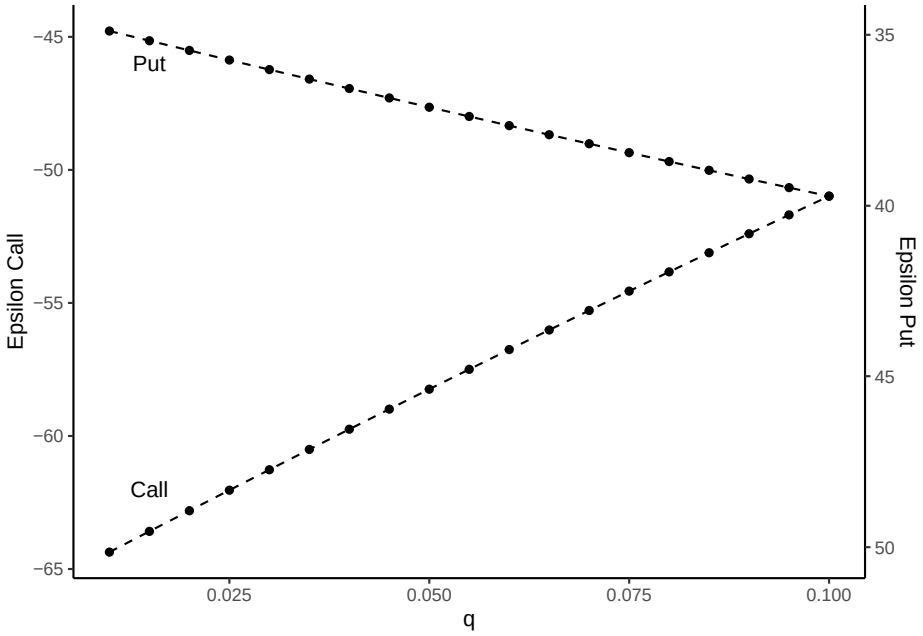

Figure 7: Option price sensitivity to the dividend yield

4 Question 3 – Extra – 3 points

4.1 Sanity check to make sure calculation is right

- American call price should equal to Europe call price: Pass
- For an American put early exercise is sometimes optimal and optimal frontier is in bottom of tree: Pass

4.2 American Digital Option

4.2.1 Arguments:

- sigma: 0.4
- rf: 0.05
- T: 1
- Nsteps: 10

4.2.2 Digital price and optimal exercising frontier

A digital option is a type of options contract that has a fixed payout if the underlying asset moves past the predetermined threshold or strike price. In our case, the payoff can be written as

$$C = \begin{cases} 1 & \text{if } S \geq K \\ 0 & \text{otherwise} \end{cases}$$

Table 11: Digital price with different inputs:

	S100|K90	S90|K100
Am|C	1.0000	0.7276
Am|P	0.7628	1.0000
Eu|C	0.5647	0.3312
Eu|P	0.3865	0.6200

Table 12: Call Am Option price with K = 100, S0 = 90

	Step1	Step2	Step3	Step4	Step5	Step6	Step7	Step8	Step9	Step10
0.73	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1	1
NA	0.47	0.70	1.00	1.00	1.00	1.00	1.00	1.00	1	1
NA	NA	0.26	0.42	0.67	1.00	1.00	1.00	1.00	1	1
NA	NA	NA	0.11	0.20	0.35	0.61	1.00	1.00	1	1
NA	NA	NA	NA	0.03	0.06	0.11	0.24	0.49	1	1
NA	NA	NA	NA	NA	0.00	0.00	0.00	0.00	0	0
NA	NA	NA	NA	NA	NA	0.00	0.00	0.00	0	0
NA	NA	NA	NA	NA	NA	NA	0.00	0.00	0	0
NA	NA	NA	NA	NA	NA	NA	NA	0.00	0	0
NA	NA	NA	NA	NA	NA	NA	NA	NA	0	0
NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	0

Table 13: Optimal exercising frontier: with K = 100, S0 = 90 (Call Am, 1 for optimal exercising timming, 0 for not exercising)

Step0	Step1	Step2	Step3	Step4	Step5	Step6	Step7	Step8	Step9	Step10
0	1	1	1	1	1	1	1	1	1	1
NA	0	0	1	1	1	1	1	1	1	1
NA	NA	0	0	0	1	1	1	1	1	1
NA	NA	NA	0	0	0	0	1	1	1	1
NA	NA	NA	NA	0	0	0	0	0	1	1
NA	NA	NA	NA	NA	0	0	0	0	0	0
NA	NA	NA	NA	NA	NA	0	0	0	0	0
NA	NA	NA	NA	NA	NA	NA	0	0	0	0
NA	NA	NA	NA	NA	NA	NA	NA	0	0	0
NA	NA	NA	NA	NA	NA	NA	NA	NA	0	0
NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	0

Table 14: Call Am Option price with K = 100, S0 = 70

	Step1	Step2	Step3	Step4	Step5	Step6	Step7	Step8	Step9	Step10
0.31	0.47	0.70	1.00	1.00	1.00	1.00	1.00	1.00	1	1
NA	0.16	0.26	0.42	0.67	1.00	1.00	1.00	1.00	1	1
NA	NA	0.06	0.11	0.20	0.35	0.61	1.00	1.00	1	1
NA	NA	NA	0.01	0.03	0.06	0.11	0.24	0.49	1	1
NA	NA	NA	NA	0.00	0.00	0.00	0.00	0.00	0	0
NA	NA	NA	NA	NA	0.00	0.00	0.00	0.00	0	0
NA	NA	NA	NA	NA	NA	0.00	0.00	0.00	0	0

	Step1	Step2	Step3	Step4	Step5	Step6	Step7	Step8	Step9	Step10
NA	NA	NA	NA	NA	NA	NA	0.00	0.00	0	0
NA	NA	NA	NA	NA	NA	NA	NA	0.00	0	0
NA	NA	NA	NA	NA	NA	NA	NA	NA	0	0
NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	0

Table 15: Optimal exercising frontier: with K = 100, S0 = 70 (Call Am, 1 for optimal exercising timming, 0 for not exercising)

	Step1	Step2	Step3	Step4	Step5	Step6	Step7	Step8	Step9	Step10
0	0	0	1	1	1	1	1	1	1	1
NA	0	0	0	0	1	1	1	1	1	1
NA	NA	0	0	0	0	0	1	1	1	1
NA	NA	NA	0	0	0	0	0	0	1	1
NA	NA	NA	NA	0	0	0	0	0	0	0
NA	NA	NA	NA	NA	0	0	0	0	0	0
NA	NA	NA	NA	NA	NA	0	0	0	0	0
NA	NA	NA	NA	NA	NA	NA	0	0	0	0
NA	NA	NA	NA	NA	NA	NA	NA	0	0	0
NA	NA	NA	NA	NA	NA	NA	NA	NA	0	0
NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	0

Note that "1" in the above Table 13 and Table 15 means exercising option at this node while "0" for not exercising option.

Question: How does the Optimal Exercising Frontier vary over time?

Answer:

If S0 > K in American call option or S0 < K in American put option, then Exercising Frontier is always at node 0; One can easily notice when from S0 = 90(Table13) to S0 = 70(Table15), the optimal frontier moves backwards because the the number of paths where St > K decrease.

Comment: It's easy to notice that if S0 > K in American call option or S0 < K in American put option, then option price would be 1 as shown in Table 11. Also notice that the price of American digital call option is larger than that in Europe option case which is not always the case in plain vanilla American option (early exercise is never optimal)

4.3 Spread option

For a spread call, the payoff can be written as $C = \max(S_1 - S_2 - K, 0)$ where S1 and S2 are the prices of the two assets and K is a constant called the strike price. For a spread put it is $P = \max(K - S_1 + S_2)$

4.3.1 Arguments for :

- S0: 50
- K: 20
- sigma: 0.4
- rf: 0.05
- T: 1
- S2: 30

- S2 sigma: 0.3
- Nsteps: 2

I have to say this American spread option price is most challenging question in this assignment and is the one I spent most time on it while it only counts 0.5 points. Until the last minute I still do not have a perfect answer.

The main difficulties are:

1. with two underlying assets, first you have to set a correlation ρ between the two GBM simulation(Multivariate Brownian Motion);
2. from former point, it would be easy to do the spread option price in Monte Carlo method rather than CRR model;
3. if we persist the CRR model, we have to make an assumption that two assets evolution of two assets prices are independent, which will help us use joint risk neutral probability instead of risk neutral probability generated by asset1 or asset 2;
4. still, binomial tree is not enough because it is two dimension. We need a three dimension tree to back forward the two joint binomial tree. Each node(lattice) in this 3 dimension tree has a state with S_{1i}, S_{2i}, K, (i for different step);
5. Coding this method is not easy and also for printing optimal exercising frontier in excel sheet.

Considering all of these difficulties, I came up with a possible solution: a quadrinomial tree(see Figure 8) which has sub-nodes and aims to solve the most simple situation($\rho = 0$). In excel to make things simple to illustrate, the output only consider American spread option with CRR tree with 2 steps(Am put, Eu call/put should be similar).The Nsteps could not be large than 10, due to $4^{10} = 1048576$, the maximum rows in excel.

I also write a python script (Appendix - American call spread optoin price in python) for solving this quadrainomial tree problem since it's not efficient for me to write VBA code.

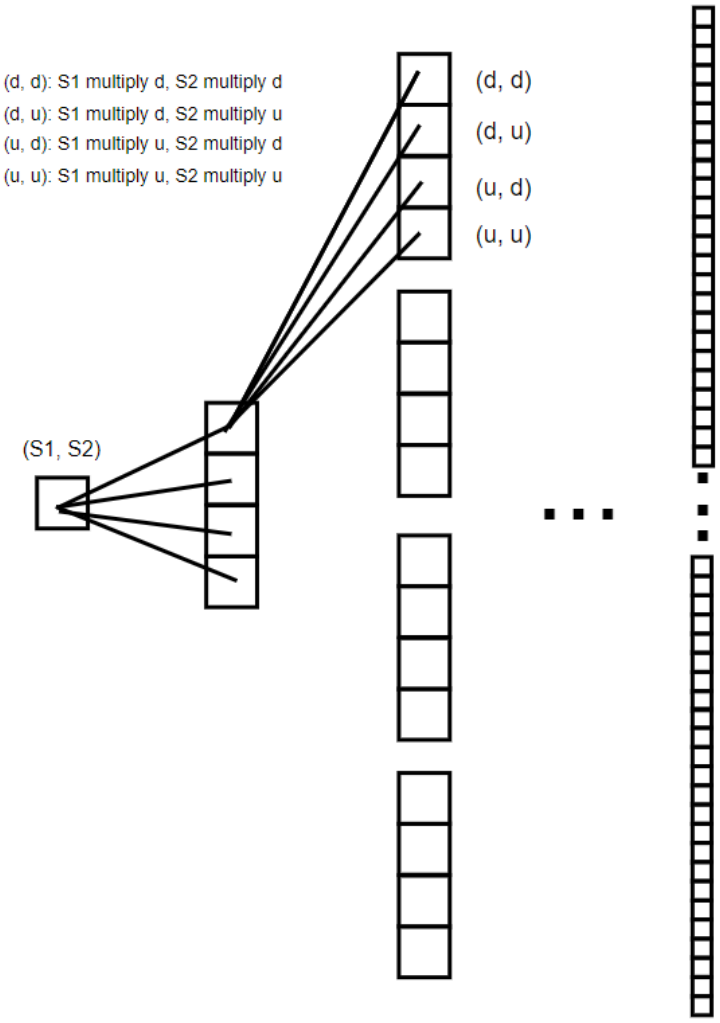

Figure 8: Illustration of quadrinomial tree

Notice:
I don't use VBA code for this part. So for this part, the parameters would not be that flexible. Also, I manually put excel formula to get the option price (seems like hard coded)

4.3.2 Result

Please see the calculation result in tab "ans_3spread".

As shown in Figure 10, the price of this option is 8.99 and I check this result with Kirk's approximation for Eu spread options price (Lo 2013), which gives the result 7.853849. Therefore, we could trust our computation process. Also, it's easy to see from Figure 11 that this call option should never be exercised before expiration as plain vanilla American call option does.

Also, running result from python script, we get the price 8.9955 for 2 steps, 9.2490 for 5 steps, 9.1226 for 10 steps, which seems our result doesn't vary too much for different steps size.

0	0	1	1	2	2
50	30	66.34482	37.08933	88.03271	45.85395
				88.03271	30
				50	45.85395
				50	30
		66.34482	24.26574	88.03271	30
				88.03271	19.62753
				50	30
				50	19.62753
		37.68192	37.08933	50	45.85395
				50	30
				28.39854	45.85395
				28.39854	30
		37.68192	24.26574	50	30
				50	19.62753
				28.39854	30
				28.39854	19.62753

Figure 9: Two stock price evoltuion

0	1	2
8.995574	13.86854	22.17875
		38.03271
		0
		0
	22.57289	38.03271
		48.40518
		0
		10.37247
	0	0
		0
		0
		0
	2.366534	0
		10.37247
		0
		0

Figure 10: Spread option value at each node

0	1	2
0	0	1
		1
		0
		0
	0	1
		1
		0
		1
	0	0
		0
		0
		0
	0	0
		1
		0
		0

Figure 11: Optimal exercising frontier at each node

5 Appendix - American call spread optoin price in python

```
import numpy as np
dd = np.array([0,0,1])
du = np.array([0,1,1])
ud = np.array([1,0,1])
uu = np.array([1,1,1])

unit = [dd,du,ud,uu]

def next_leaf(lastNode):
    res = []
    for leaf in lastNode:
        for ele in unit:
            res.append(leaf + ele)
    return res

def recursive_node(n):
    node = []
    if n == 0:
        node.append(np.array([0,0,0]))
        return node
    else:
        return next_leaf(recursive_node(n-1))

def price(node, s1, s2, u1, d1, u2, d2):
    res = [] # store pairs of s1, s2 price at node
```

```
    for leaf in node:
        i,j,n = leaf[0],leaf[1],leaf[2]
        price = (s1 * u1**i * d1**(n-i), s2 * u2**j * d2**(n-j), n)
        res.append(price)
    return res

def payoff(pairPrice, K):
    res = []
    for pair in pairPrice:
        res.append(max(pair[0] - pair[1] - K, 0))
    return res

def chunks(lst, n):
    """Yield successive n-sized chunks from lst."""
    for i in range(0, len(lst), n):
        yield lst[i:i + n]

def value(T, n, sigma1, sigma2, rf, s1, s2, K):
    dt = T/n
    u1 = np.exp(sigma1 * np.sqrt(dt))
    d1 = 1/u1
    p1 = (np.exp(rf * dt) -d1)/(u1 - d1)
    u2 = np.exp(sigma2 * np.sqrt(dt))
    d2 = 1/u2
    p2 = (np.exp(rf * dt) -d2)/(u2 - d2)
    discount = np.exp(-rf * dt)
    prob = np.array([(1-p1)*(1-p2), (1-p1)*p2, p1*(1-p2), p1*p2])

    res = [0.] * (n + 1) # create list to store backward option value in each step
    payoffTree = [0.] * (n + 1)
    for i in range(n + 1):
        payoffTree[i] = payoff(price(recursive_node(i), s1, s2, u1, d1, u2, d2), K)
    res[n] = payoffTree[n]

    for step in reversed(range(n)):
        temp = []
        for i, leaf in enumerate(chunks(res[step+1],4)):
            temp.append(max(payoffTree[step][i], np.inner(leaf, prob) * discount))
        res[step] = temp
    return res[0]
```

6 Appendix – Financial Functions

```
ValueAtRisk(retArr As Variant, confidence As Double)
MertonRateMCsim(startValue As Double, mu As Double, sigma As Double, T As Double,Nsteps
As Integer, nSim As Integer)
StockPriceSim_NextOneStep(startValue As Double, ByVal rf As Double, sigma As Double, T As
Double)
StockPriceMCsim(startValue As Double, rf As Double, sigma As Double, T As Double, Nsteps
As Integer, nSim As Integer)
StockPriceMCsim(startValue As Double, rf As Double, sigma As Double, T As Double, Nsteps
As Integer, nSim As Integer)
```

```
SelectRatePatterns(startValue As Double, mu As Double, sigma As Double, T As Double,
Nsteps As Integer, nSim As Integer)
StockPriceMCsim_withMertonRate(startValue As Double, mu As Double, sigma As Double, T As
Double, Nsteps As Integer, nSim As Integer, S0 As Double, stockSigma As Double, StocknSim
As Integer)
payoff(K As Double, ByVal ST As Double, iscall As Boolean, Optional isdigital As Boolean
= False, Optional isSpread As Boolean = False, Optional ByVal S2 As Double)
BSM(S As Double, K As Double, vol As Double, rf As Double, T As Double, q As Double,
iscall As Boolean)
GreeksBSM(S As Double, K As Double, vol As Double, rf As Double, T As Double, q As Double,
numPoints As Integer, Optional precision As Double = 100)
CRR_Model(S0 As Double, K As Double, vol As Double, rf As Double, T As Double, Nsteps As
Integer, iscall As Boolean, isAmerican As Boolean, Optional isdigital As Boolean = False,
Optional isSpread As Boolean = False, Optional S2 As Double)
```

References

Lo, Chi-Fai. 2013. "A Simple Derivation of Kirk's Approximation for Spread Options." *Applied Mathematics Letters* 26 (8): 904–7.