CQF Module 2 Exam Report

Candidate Name: Siqi He

March 2023

Question 1.

The calculations and supporting Python code are presented in the attached Jupyter Notebook "Question 1.ipynb".

(a) The Lagrangian function L(x, λ) is our objective function *f* augmented by the addition of the constraint functions, where each function is augmented by the Lagrangian multiplier λ.

In case of the minimum variance portfolio (as described in the question) we can formulate the optimisation framework:

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} \frac{1}{2}\boldsymbol{w}'\boldsymbol{\Sigma}\boldsymbol{w}$$

Subject to:

$$\text{s.t. } \boldsymbol{w}'\mathbf{1} = 1$$
$$\mu_\Pi = \boldsymbol{w}'\boldsymbol{\mu} = m$$

The Lagrangian function can then be formulated as:

$$L(\varpi, \lambda, \gamma) = \frac{1}{2}\varpi'\Sigma\varpi + \lambda(m - \varpi'\mu) + \gamma(1 - \varpi'\mathbf{1})$$

The partial derivatives are computed as part of setting the first order condition to zero.

$$\frac{\partial L}{\partial \varpi} = \frac{1}{2} \times 2 \times \Sigma\varpi + \lambda(-\mu) + \gamma(-\mathbf{1}) = \Sigma\varpi - \lambda\mu - \gamma\mathbf{1}$$

$$\frac{\partial L}{\partial \lambda} = m - \varpi'\mu$$

$$\frac{\partial L}{\partial \gamma} = 1 - \varpi'\mathbf{1}$$

(The second order derivative $\frac{\partial^2 L}{\partial \varpi^2} = \Sigma$ which is the covariance matrix, a positive indefinite)

(b) The analytical solution of ϖ^*:

$$\varpi^* = \frac{1}{AC+B^2}\Sigma^{-1}[(A\mu - B\mathbf{1})m + (C\mathbf{1} - B\mu)] \qquad (1)$$

Where scalars A, B C are:

$$A = \mathbf{1}'\Sigma^{-1}\mathbf{1}$$
$$B = \mu'\Sigma^{-1}\mathbf{1} = \mathbf{1}'\Sigma^{-1}\mu$$
$$C = \mu'\Sigma^{-1}\mu$$

Substituting the values of A, B, C into (1) we get:

$$\varpi^* = \frac{1}{(\mathbf{1}'\Sigma^{-1}\mathbf{1})(\mu'\Sigma^{-1}\mu)+(\mathbf{1}'\Sigma^{-1}\mu)^2}\Sigma^{-1}[(\mathbf{1}'\Sigma^{-1}\mathbf{1}\mu - \mathbf{1}'\Sigma^{-1}\mu\mathbf{1})m + (\mu'\Sigma^{-1}\mu\mathbf{1} - \mathbf{1}'\Sigma^{-1}\mu\mu)]$$

(c) The inverse optimisation of 700 random allocations is computed in the attached Jupyter Notebook.

The "*portfolio_simulations*" function takes in the array of expected returns (0.02, 0.07, 0.15, 0.20) as given for assets in the portfolio and returns a data frame, with each observation representing a portfolio weight allocation, and columns storing (in sequential order) the portfolio returns, portfolio volatility, portfolio weight vector, and the Sharpe Ratio.

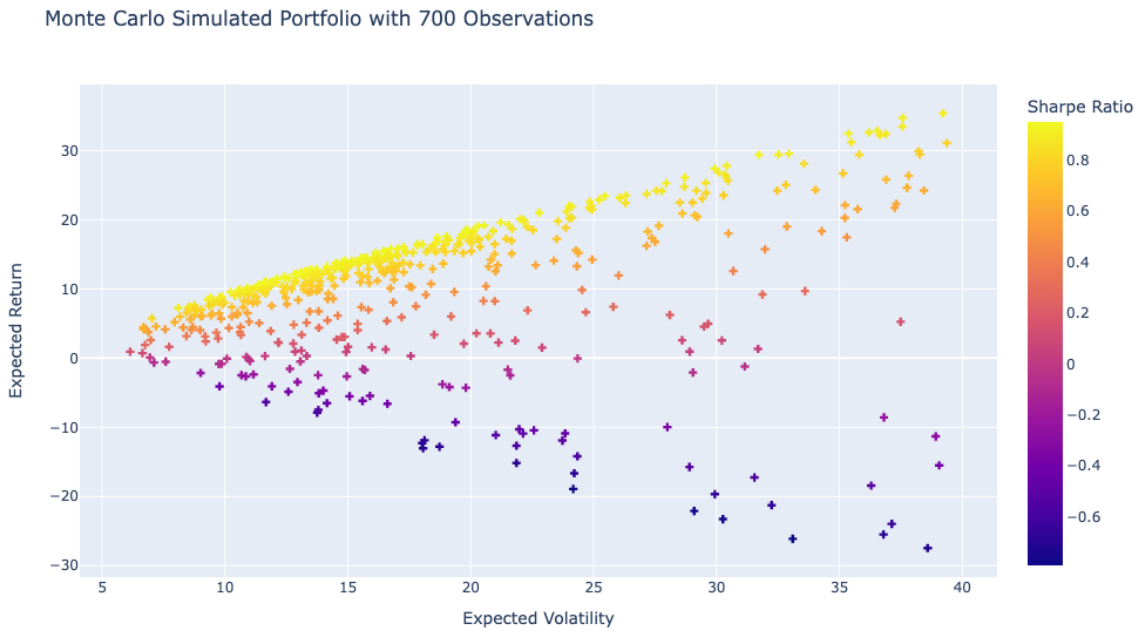
Known parameters, *sigma* (vector of volatility for each individual asset transformed in to a diagonal matrix), *numofasset* (the number of assets in the portfolio), *numofportfolio* (the number of simulated portfolios i.e. randomly generated weight allocations) as given in the requirements.

The covariance matrix Σ is computed by:

$$\Sigma = SRS$$

Where R is the correlation matrix as given in the question. And S is the diagonal volatility matrix.

The portfolio volatility and portfolio return columns are extracted from the output data frame and plotted as below (extracted from the Jupyter Notebook attached. All dots are colour coded according to the corresponding Sharpe Ratio, where the lighter the colour the larger the Sharpe Ratio. Note that upper bounds are manually set for the X-axis and Y-axis values to avoid distortion of graph scale from large outliers. The snapshot below has upper bounds of 80 for the X-axis and 40 for the Y-axis and should capture majority of the dots plotted after trying various combination of bounds.



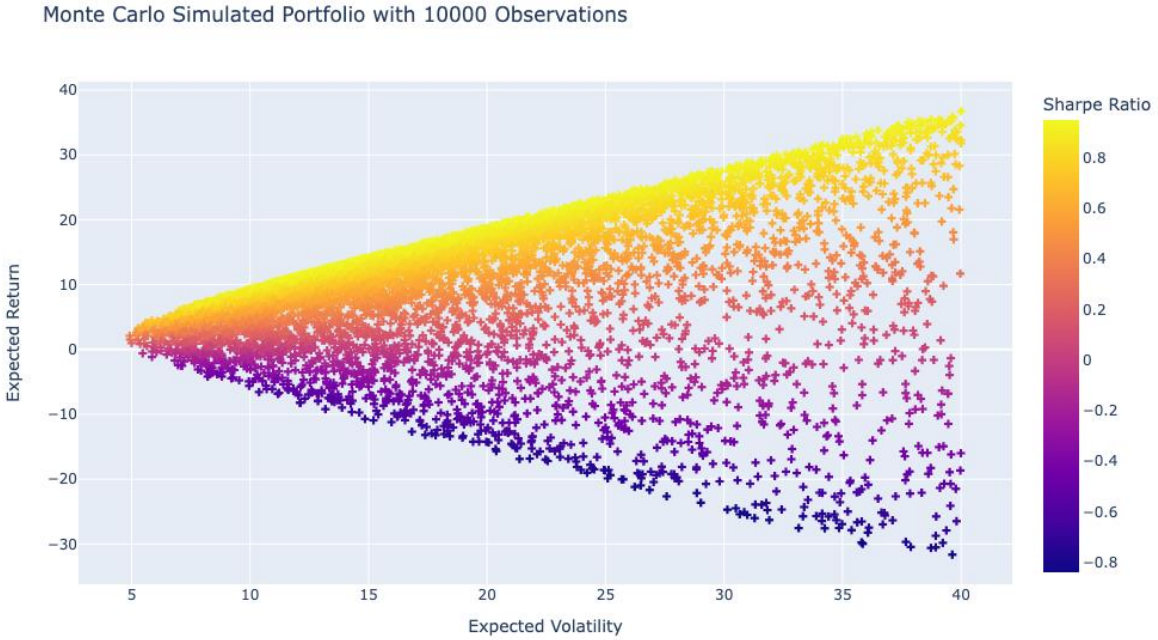
Explanation of the plot:

On this plot every dot represents a portfolio allocation between the 4 risky assets specified in the question. Somewhere along the upper edge of the scatter where the dots are shown in bright yellow is a simulated approximation of the efficient frontier (approximation because we have only simulated 700 allocations hence not capturing all possible portfolios allocations). Note that in the simulation we assume that we can either long or short the assets as long as 4 assets combined makes up the entirety of the portfolio. This means that some of the simulations will have negative weights allocated to one or more asset.

As presented in colour coding, as the Sharpe Ratio is higher as dots become closer to the yellow edge. The Sharpe Ratio for a portfolio of risky assets (without a risk-free asset) is defined as the expected portfolio return divided by the expected portfolio volatility (μ_Π/σ_Π). Therefore, the higher the Sharpe Ratio, the better the portfolio since it indicates that the return is higher per unit of volatility.

Hence it is no coincidence that the allocations close to our approximated efficient frontier have highest bands of Sharpe Ratios, as the efficient frontier is defined as the set of optimal portfolios that offer the highest expected return for a given level of risk, or the lowest risk for a given level of expected return. Portfolios that lie below the efficient frontier are sub-optimal because they do not provide the highest possible return for the level of risk. Here amongst our 700 simulated portfolios, the allocations along the yellow-coloured edge form the set of optimal portfolios that are ideal for investors whilst the ones below in darker colours are sub-optimal and ought not be invested in.

As an additional note, a better approximation of the efficient frontier for this pool of the given four assets can be computed by increasing the number of simulations. As the simulated portfolios increase, the scattered plots should become gradually closer to forming a hyperbolic plane with a clear (and more solid) yellow edge forming the efficient frontier. For illustration purposes, the below graph is generated using 10000 simulated weights.



Question 2.

The calculations and supporting Python code are presented in the attached Jupyter Notebook "Question 2.ipynb".

(a) For a tangency portfolio with a risk-free asset, to solve for the minimum variance portfolio with a target return the optimisation problem becomes:

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} \frac{1}{2}\boldsymbol{w}'\boldsymbol{\Sigma}\boldsymbol{w}$$

Subject to:

$$r + \varpi'(\mu - r\mathbf{1}) = m$$

The budget constraint is no longer present because the residual of the wealth not invested in risky assets is not in the risk-free asset.

(b) The Lagrange function can therefore be formulated as:

$$L(\varpi, \lambda) = \frac{1}{2}\varpi'\Sigma\varpi + \lambda[m - r - \varpi'(\mu - r\mathbf{1})]$$

For which the partial derivatives are:

$$\frac{\partial L}{\partial \varpi} = \frac{1}{2} \times 2 \times \Sigma\varpi - \lambda(\mu - r\mathbf{1}) = \Sigma\varpi - \lambda(\mu - r\mathbf{1})$$

$$\frac{\partial L}{\partial \lambda} = m - r - \varpi'(\mu - r\mathbf{1})$$

(The second order derivative $\frac{\partial^2 L}{\partial \varpi^2} = \Sigma$ which is the covariance matrix, a positive indefinite)

(c) The tangency portfolio is the portfolio that is entirely invested in risky assets and is on the CAPM Line.

The ready formulas for the tangency portfolio statistics are given in Slide 86 and 88 in Module 2 Lecture 2:

$$\varpi_t = \frac{\Sigma^{-1}(\mu - r\mathbf{1})}{B - Ar}$$

$$\sigma_t = \sqrt{\varpi'_t \Sigma \varpi_t} = \sqrt{\frac{C - 2rB + r^2 A}{(B - Ar)^2}}$$

Where scalars A, B and C are:

$$A = \mathbf{1}'\Sigma^{-1}\mathbf{1}$$
$$B = \mu'\Sigma^{-1}\mathbf{1}$$
$$C = \mu'\Sigma^{-1}\mu$$

The calculations are performed in the Python code attached.

The risk-free rate (r= 0.005, 0.01, 0.015, 0.0175 respectively), vector of returns (μ= [0.02, 0.07, 0.15, 0.20]), vector of volatilities (σ=[0.05, 0.12, 0.17, 0.25]), are all given in the question as known parameters.

The covariance matrix Σ is computed by:

$$\Sigma = SRS$$

Where R is the correlation matrix as given in the question. And S is the diagonal volatility matrix.

Plugging in values of known parameters, we compute the values of A, B and C:

A= 423.61498069
B= 6.80702072
C= 0.90650893

The *tangent_opt_stats* function is then defined to take in the risk-free rate *r* as an argument and output 4 portfolio statistics: weights, volatility, returns and slope of the CAPM line.

The 4 risk-free rates given are then respectively inputted into the function, and the *output_df* table stores the portfolio statistics for each of risk-free rate.

Results are presented below:

Risk Free Rate	Optimal Allocation (corresponding in order to the four assets A, B, C and D)	(σ_t rounded to 4 decimals)
0.005	0.0168352, -0.22936698, 0.81434026, 0.39819152	0.1965
0.010	-0.74593711, -0.51056937, 1.49024934, 0.76625714	0.3507
0.015	-8.64485405, -3.42257114, 8.48965087, 4.57777433	1.9724
0.0175	8.10350247, 2.75185052, -6.3514309 , -3.50392209	1.4735

<u>Plotting the efficient frontier for r_f = 100bps, 175 bps</u>

In the presence of a risk-free asset, the new efficient fronter becomes the Capital Market line, which is a line that is at a tangency (shares one point) with the previous hyperbolic efficient frontier for portfolios with only risky assets, because for all levels of risk, this is the line that delivers the highest return compared to all other possible portfolios.

The slope of the CAPM line is the Sharpe Ratio, which is calculated by:

$$S_t = \frac{\mu_t - R}{\sigma_t}$$

And its intercept with the Y-axis is the risk-free rate.

In the Python code, the Sharpe Ratio has already been calculated in the *tangency_portfolio_stats* function and stored in the *output_df* table. Shown below:

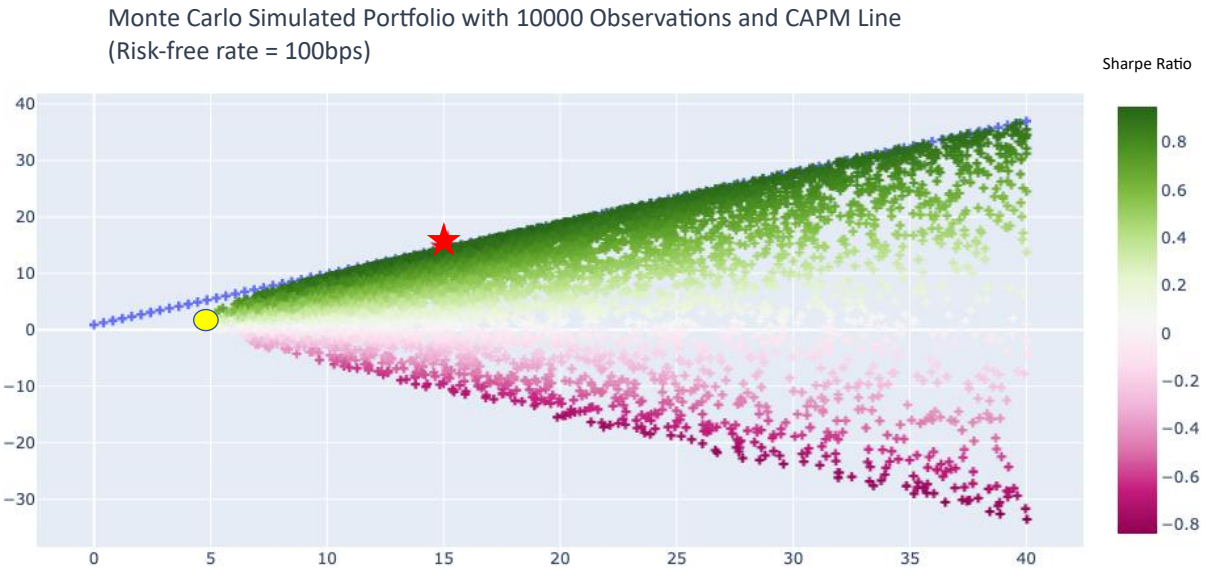
Risk Free Rate (Intercept)	Sharpe Ratio (Slope)
0.0100	0.9015
0.0175	-0.8933

Now knowing the slope and intercept, we plot the two lines as below:

Here everything looks fine, but surely there is something wrong about the plot below when the risk-free rate is 175bps!

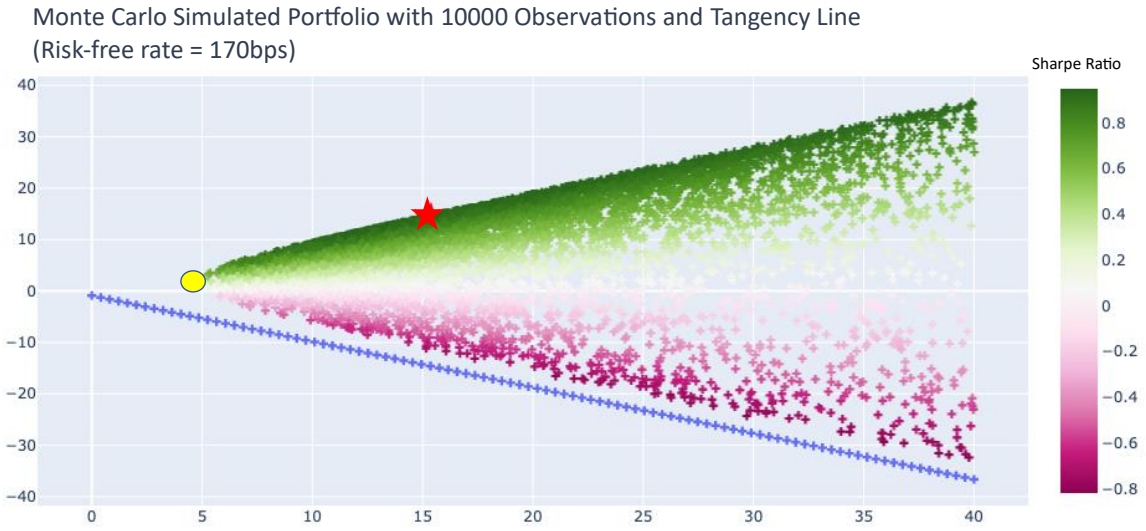
It makes no sense that the slope is negative... To investigate, I have further plotted these lines alongside the simulated risky-asset portfolios in Question 1. All computations are performed in the Python Script attached.

When the risk-free rate is 100bps, the plot looks like so:



The plot makes sense as we can see the tangency portfolio (represented by the red star) is on the upper left quadrant of the graph, where returns are highest for a given level of risk.

However, plotting the line when the risk-free rate is 175bps, we see that the line is not at a tangency with the upper left quadrant of the graph. In fact (although not clearly seen as the edge of the hyperbola is not clearly formulated by the scatter plot), the point that is shared between the computed "CAPM" line and the hyperbola has a negative return! It makes no sense that when a risk-free rate is 175bps, the optimal portfolio produces loss.



Without going into excessive detail, this "imposter" CAPM line is computed as such because the risk-free rate of 175bps is already higher than the returns on the hyperbolic plane when the risk is at its lowest (represented by the yellow-coloured dot marked on the graph). In the Python code, we have calculated that this benchmark return is approximately 145bps.

This means that with a risk-free rate of 175bps whatever line you draw from the intercept, it cannot be at a tangency with the upper edge of the hyperbola where returns are highest for the risky asset portfolio (at some point in the far fight of the graph it will eventually intersect with

the hyperbola). Hence the algorithms rendered that only a negative slope makes possible to form a tangency with the lower edge of the hyperbola.

In reality, if a risk-free rate is already higher than the risky portfolio returns with the lowest possible risk, any investor (unless they have some very unusual reason to take on the leverage) would simply put all their money into the risk-free asset, since it is by definition a risk-free return. Any incorporation of risky asset would mean taking on disproportionally higher risk for a very small increase in portfolio returns.

Therefore, whilst we can find a tangency line and a tangency portfolio, it is by nature not the efficient frontier of a portfolio with risk-free asset, and the tangency point (or any point on the tangency line) certainly does not represent an optimal allocation (or highest return portfolio at a given level of risk). I would say that the optimal portfolio allocation in this scenario is to simply invest 100% of the money in the risk-free asset.

Question 3.

The calculations and supporting Python code are presented in the attached Jupyter Notebook "Question 3.ipynb".

(a) Parameterisation for up and down moves are chosen as:

$$uS = 1 + \sigma\sqrt{t}$$
$$vS = 1 - \sigma\sqrt{t}$$

(b) A function "*binomial_option*" is defined to calculate the binomial option price. It takes in 7 parameters *spot*, *strike*, *rate*, *sigma*, *time*, *steps*, *output* (with default value of 2 which selects the option price element in the output array).

Due to continuous compounding, the discount factor ("df") is expressed as:

$$\frac{1}{e^{-rt}}$$

Where $\mathbf{r}$ is the interest free rate 0.05 and t is the time interval (=time/steps=1/4)

The parameters are assigned values as per instructions: *spot*=100, *strike*=100, *rate*=0.05, *time*=1, *steps*=4. The program then loops through the range of volatilities, assigning them to the sigma parameter, to compute a price for each option. The price array is indexed at [0,0] to extract the price at time zero.

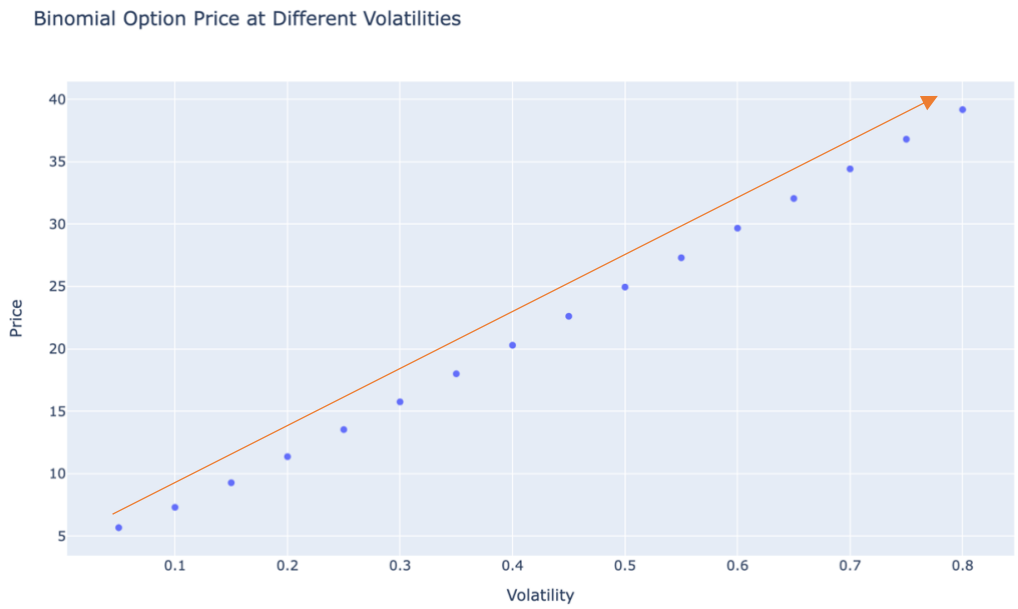
The calculated range of option price presented here (extracted from Jupyter Notebook outputs):

Step	4.00	5.00	6.00	7.00	8.00	9.0	10.00	11.00	12.00	13.00	14.00	15.00	16.00	17.00	18.00	19.0	20.00	21.00	22.00
Price	11.36	11.85	11.47	11.75	11.52	11.7	11.54	11.66	11.56	11.64	11.57	11.62	11.57	11.61	11.58	11.6	11.58	11.59	11.58

Step	23.00	24.00	25.00	26.00	27.00	28.00	29.00	30.00	31.00	32.00	33.00	34.00	35.00	36.00	37.00	38.00	39.00	40.00
Price	11.58	11.58	11.58	11.58	11.57	11.58	11.57	11.58	11.56	11.58	11.56	11.58	11.56	11.58	11.56	11.58	11.55	11.58

Step	40.00	41.00	42.00	43.00	44.00	45.00	46.00	47.00	48.00	49.00	50.00
Price	11.58	11.55	11.58	11.55	11.58	11.55	11.58	11.55	11.58	11.54	11.58

Plotted as (extracted from Jupyter Notebook outputs):



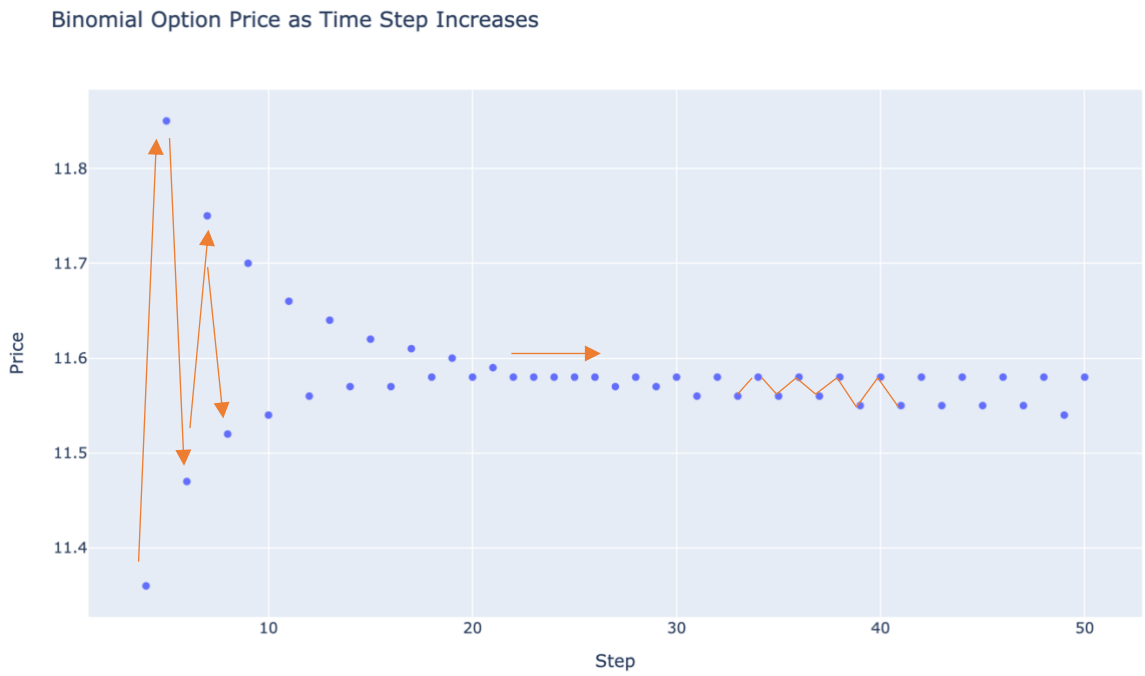
The plot shows that as volatility goes up the price of the option increases. This is because for an option you cannot lose money (as the lowest possible payoff is zero). Therefore, theoretically the higher the volatility the better chances of making a larger profit.

(c) The parameters are then assigned values as per instructions: *spot*=100, *strike*=100, *rate*=0.05, *sigma*=0.20, *time*=1. The program then loops through the range of time steps from 4 to 50, assigning them to the *steps* parameter, to compute a price for each option. The price array is indexed at [0,0] to extract the price at time zero.

The calculated range of option price presented here (extracted from Jupyter Notebook outputs):

Step	4.00	5.00	6.00	7.00	8.00	9.0	10.00	11.00	12.00	13.00	...	41.00	42.00	43.00	44.00	45.00	46.00	47.00	48.00	49.00	50.00
Price	11.36	11.85	11.47	11.75	11.52	11.7	11.54	11.66	11.56	11.64	...	11.55	11.58	11.55	11.58	11.55	11.58	11.55	11.58	11.54	11.58

Plotted as (extracted from Jupyter Notebook outputs):



The plot shows as time step increases, the price of the option shows a trend of convergence from initial high-low jumps. While convergence seem to materialise when time step = 20, from this graph alone we seem to see a slight divergence between step = 20 to step = 20.

However, on an additional note, upon further investigation, it seems to that the reason convergence does not appear stable is because the maximum time steps (set at 50) not being large enough to fully capture the trend. Once we increase the time step to higher numbers (e.g., 500) we can clearly see that the price converges to approximately 11.55. This idea is explored in the Jupyter Notebook attached.

Question 4.

The calculations and supporting Python code are presented in the attached Jupyter Notebook "Question 4.ipynb".

The ready formula for Expected Shortfall is given as:

$$\mathrm{ES}_c(X) = \mu - \sigma \frac{\phi(\Phi^{-1}(1-c))}{1-c}.$$

As we are calculating the standardised ES for N(0,1), we can infer:

$$\mu = 0$$
$$\sigma = 1$$

c denotes the confidence levels, which are given in the question as [99.95, 99.75, 99.5, 99.25, 99, 98.5, 98, 97.5].

Substituting individual values of c respectively into the formula, we can compute the ES for each confidence level. In the Python script, the function *ES_calc* takes in the confidence level as a parameter and outputs the corresponding standard ES.

Final results are as below:

Note that here ES is computed in terms of returns hence the negative signs. Results are rounded to two decimal places.

Confidence Level	Standardised Expected Shortfall for N(0,1)
99.95	-3.55
99.75	- 3.10
99.50	-2.89
99.25	-2.76
99.00	-2.67
98.50	-2.52
98.00	-2.42
97.50	-2.34

Question 5.

The calculations, breach counts and plots are all computed in sheet "Question 5" in the spreadsheet "Question 5 and 6.xlsx" attached in the zip folder.

(a) The formula for analytical VaR is the

*21D rolling standard deviation * sqrt(T) * factor.*

Where T=10, factor=normsinv(1-C), 21D rolling σ is calculate with log returns.

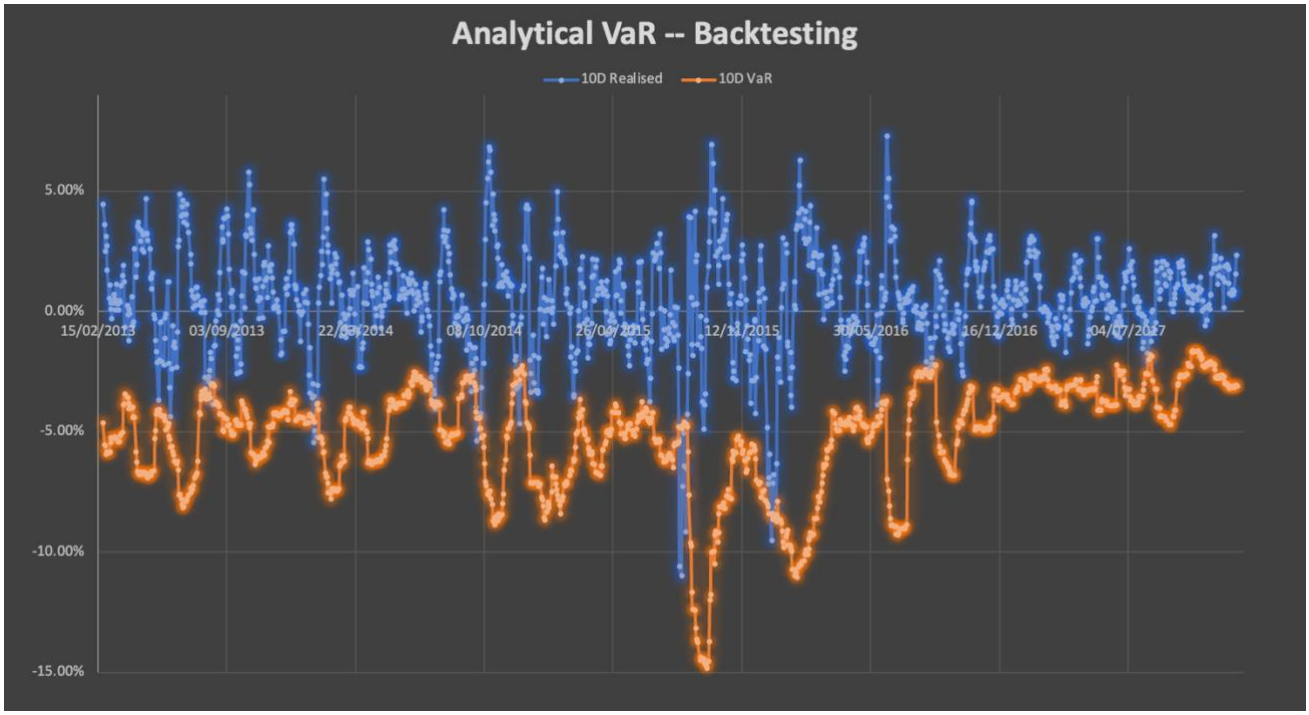
The VaR is calculated in column E (E31:E1259). Breaches are distinguished by comparing values of the calculated VaR with 10D forward realised return computed in column F (F31:F1258), at points where the realised return falls below the calculated analytical VaR.

The total number of breaches are counted (breach flag = 1 in cells G31:G1258) and presented in cell H17 as 25 breaches. Percentage of breaches is calculated by dividing the number of breaches by the total number of observed 10D forward realised returns (25 breaches/1218 total observations), presented in cell F17 as 2.05% (rounded to 2 decimal place).

(b) Consecutive breaches are calculated by taking advantage of the binary property of the breach flag in column G31:G1258. Values for each observation is multiplied by the previous, and a consecutive breach flag occurs when the product = 1, indicating that the breach flag had a value of 1 both in the current day and the day before.

The total number of consecutive breach occurrences are computed by summing across the consecutive breach flag column (H32:H1258), presented in cell H18 as 14 consecutive breaches.

(c) The graph plots the 10D VaR (E31:E1259) against the 10D realised returns (F31:F1258). The plot below is copy and pasted from the attached spreadsheet.



Question 6.

The calculations, breach counts and plots are all computed in sheet "Question 6" in the spreadsheet "Question 5 and 6.xlsx" attached in the zip folder.

(a) The 10D EWMA VaR is calculated in column H (H16:H1264).

Variance estimate is initialised by calculating mean squared returns across cells D16:D1264, which is 0.0056% rounded to four decimals presented in cell C5. Recurrent updates begin at cell D17 using the formula $\sigma^2_{t+1|t} = \lambda\sigma^2_{t|t-1} + (1-\lambda)r^2_t$.

The 1D Standard Deviation estimate (F16:F1264) is derived by taking the square root of the variance estimates. The T-Days (10 days) ahead Standard Deviation (G16:G1264) estimate is derived by scaling the 1D Standard Deviation estimates by square root T. Finally, the 10D EWMA VaR is the product of the 10D Standard Deviation estimate and the VaR factor of -2.326 computed in cell H6 from the given confidence level of 99%.

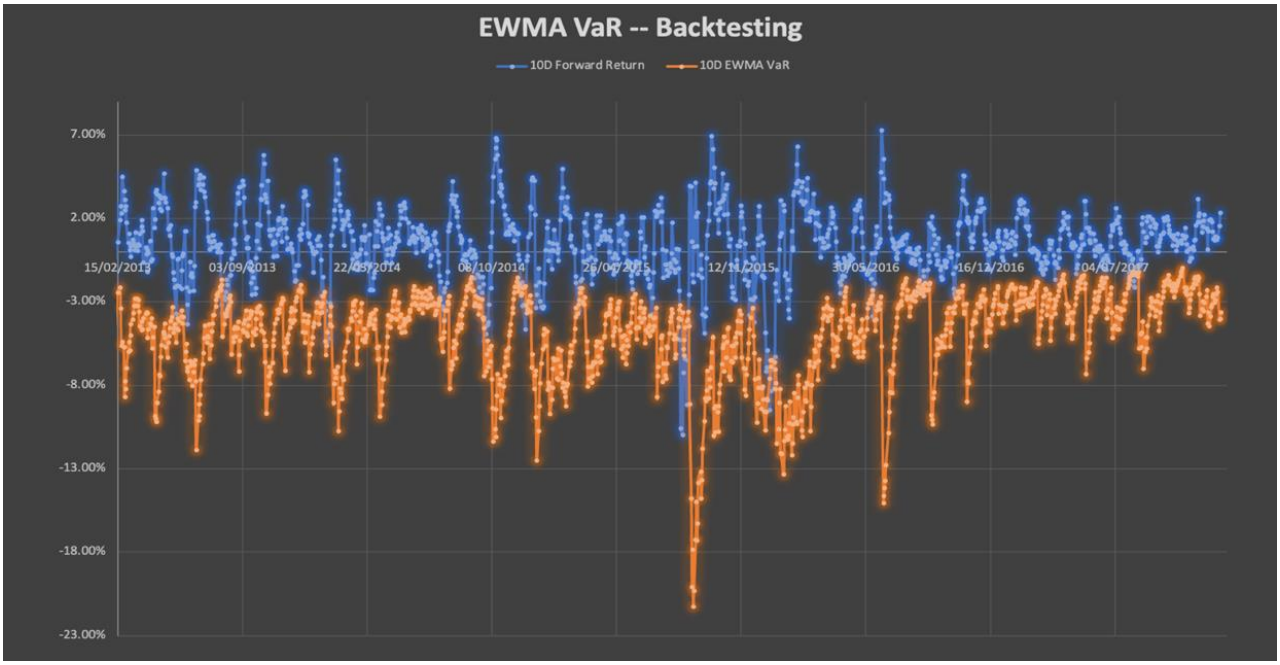
Breaches are distinguished by comparing values of the calculated EWMA VaR with 10D forward realised return computed in column I (I16:F1253), where the realised return falls below the EWMA VaR.

The total number of breaches are counted (breach flag = 1 in cells J16:J1253) and presented in cell J11 as 36 breaches. Percentage of breaches is calculated by dividing the number of breaches by the total number of observed 10D forward realised returns (36 breaches/1238 total observations), presented in cell H11 as 2.91% (rounded to 2 decimal place).

(b) Consecutive breaches are calculated by taking advantage of the binary property of the breach flag in column range J16:J1253. Values for each observation is multiplied by the previous, and a consecutive breach flag occurs when the product = 1, indicating that the breach flag had a value of 1 both in the current day and the day before.

The total number of consecutive breach occurrences are computed by summing across the consecutive breach flat column (K17:K1253), presented in cell J12 as 19 consecutive breaches.

(c) The graph plots the 10D VaR (H16:H1264) against the 10D realised returns (I16:F1253). The plot below is copy and pasted from the attached spreadsheet.



END OF REPORT