COMP6212 Computational Finance 2017/18, Assignment Part I

(a)

1. Both of the assets have the same expected return (e.g. mean) and variance. This means that any holdings combination will result in the same E and V pair and consequently, they will map into the same unique Markowitz efficient frontier.

2. The following figures, which were plotted using MATLAB's financial toolbox, show the Markowitz efficient frontier for different combinations of securities. The function used to produce them was `portstats`, using the Expected Return vector and the Covariance matrix of the securities as the core arguments.

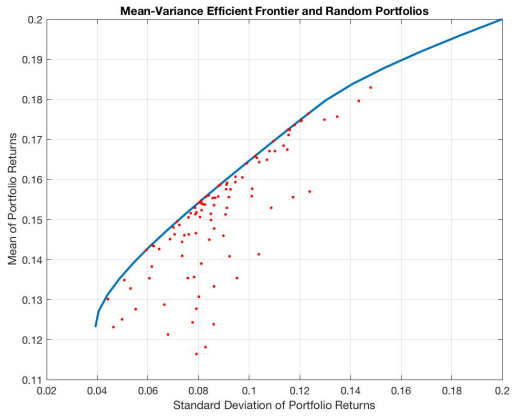

Figure 1: Efficient Portfolio for the 3 Securities

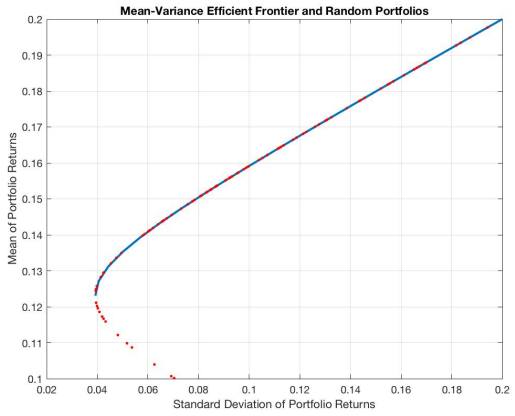

Figure 2: Efficient Portfolio of Securities 1 and 2

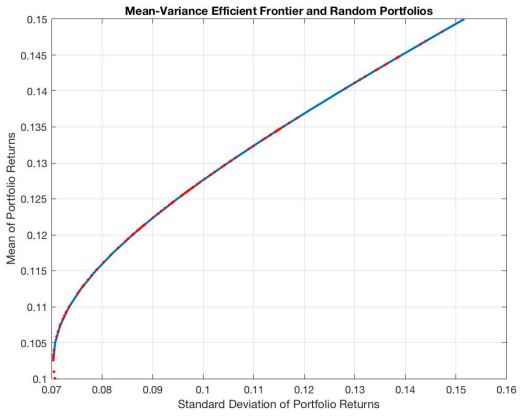

Figure 3: Efficient Portfolio of Securities 1 and 3

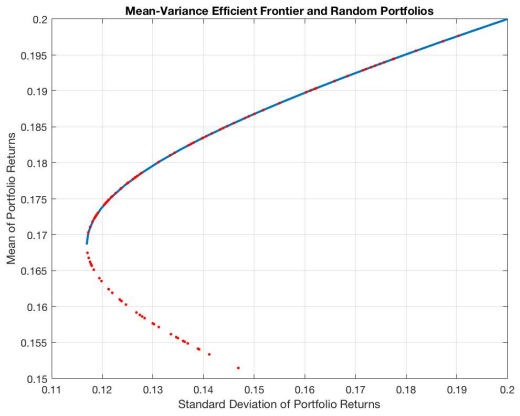

Figure 4: Efficient Portfolio of Securities 2 and 3

Figure 1 shows the Markowitz efficient frontier of the three securities with a scatter of 100 random portfolios. The random portfolios cover a wide area of the plot, showing examples of good (e.g. close to the frontier) and bad (e.g. far from the frontier) portfolios. Distinctively, Figure 2 to Figure 4 show that all of the randomly generated portfolios are positioned on the efficient frontier. This implies that holding any pair combination of these 3 securities leads to an efficient portfolio.

3. Matlab's CVX tools were used to replace the built-in `linprog` and `quadprog` functions within the **NaiveMV** function.

```
cvx_begin
    variable weights(3)
    maximize( ERet * weights )
    subject to
        V1 * weights == 1; % equality constraint, where V1 = ones(3,1)
        weights >= 0; % inequality constraint (Lower Bound)
cvx_end
```

Code 1: `linprog` in CVX

Code 1 shows the procedure in which CVX can be used to find the weights (e.g. holdings) of the securities that maximize the expected return, subject to the following conditions:

$$\sum_{i=1}^{n} \omega_i = 1 \qquad (1) \qquad\qquad \omega_i \geq 0 \qquad (2)$$

Code 2 shows the CVX implementation for the `quadprog` function. This time, the task involves finding the weights that minimize the variance of the portfolio, $\omega^T \Sigma \omega$, subject to the same constraints (1) and (2).

```
cvx_begin
    variable weights(3)
    minimize( weights' * ECov * weights )
    subject to
        V1 * weights == 1; % equality constraint, where V1 = ones(3,1)
        weights >= 0; % inequality constraint (Lower Bound)
cvx_end
```

Code 2: `quadprog` in CVX

Lastly, using the information collected from Code 1 and Code 2, that is, the maximum achievable return and the minimum achievable variance, Code 3 is used to iteratively solve the optimization problem by discretely incrementing the desired expected return to trace the efficient frontier.

```
cvx_begin
    variable weights(3)
    minimize( weights' * ECov * weights )
    subject to
        A * weights == B;%equality constraint using several target returns
        weights >= 0; % inequality constraint (Lower Bound)
cvx_end
```

Code 3: `quadprog` in CVX for several target returns

This new implementation of **NaiveMV** using the CVX tools was tested by plotting the Markowitz efficient frontier for the same securities in Question 2, against its previous version (e.g. using `linprog` and `quadprog`), as shown in Figure 5. The plot shows that both approaches lead to the same result.

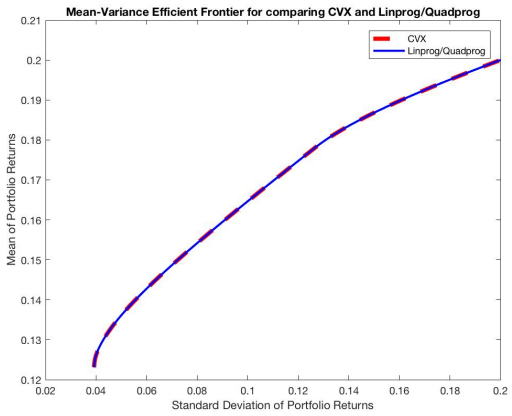

Figure 5: Comparison of Efficient Frontier using NaiveMV with CVX and `linprog/quadprog`

4. The three randomly selected stocks from the constituent companies of the FTSE 100 were: Ashtead Group (ASH), Croda International (CRDA) and Sage Group (SGE). The daily closing prices of the stocks for the past three years were first imported from Yahoo Finance into a CSV file. The data from the first year and half (e.g. half of the data), was used to compute the expected returns and covariance of the securities. The daily returns were calculated by using the following equation:

$$return_t = (closingPrice_t - closingPrice_{t-1}) \,/\, closingPrice_t \qquad (3)$$

The resulting expected return vector and covariance matrix for the securities ASH, CRDA and SGE, in that order, were:

$$\boldsymbol{m} = \begin{bmatrix} 4.045e^{-4} \\ 4.983e^{-4} \\ 1.186e^{-3} \end{bmatrix} \qquad \boldsymbol{C} = \begin{bmatrix} 4.769e^{-4} & 1.119e^{-4} & 1.135e^{-4} \\ 1.119e^{-4} & 1.747e^{-4} & 9.146e^{-4} \\ 1.135e^{-4} & 9.146e^{-4} & 2.510e^{-4} \end{bmatrix}$$

Using this information, the efficient frontier, shown in Figure 6, was then plotted using MATLAB's financial tools.

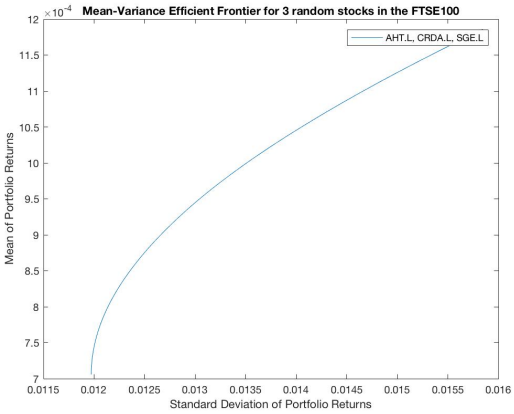

Figure 6: Efficient Frontier for ASH, CRDA and SGE

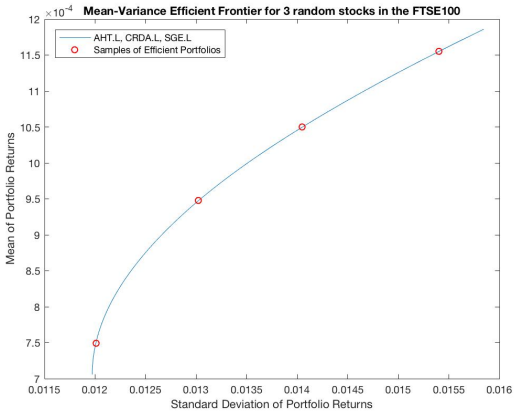

Figure 7: Four Efficient Portfolio Samples

To compare the performance of performance of several efficient portfolios obtained by using the information in the first half of the data, four portfolios, represented as red dots in Figure 7, were randomly selected to be tested using the remaining half of the data against the naïve portfolio. The obtained results are shown in the table below.

Weights =>	AHT	CRDA	SGE	Return (%)
Naive	0.3333	0.3333	0.3333	32.08
Portfolio 1	0.0000	0.0494	0.9506	-3.07
Portfolio 2	0.0000	0.1970	0.8025	3.04
Portfolio 3	0.0003	0.3453	0.6544	9.20
Portfolio 4	0.0682	0.5572	0.3746	22.73

Table 1: Returns of sample portfolios from the efficient frontier against the naïve portfolio

The results show that the returns generated from the naïve portfolio outperform the rest. The best performing random portfolio is Portfolio 4, which is the portfolio with the highest mean of return (e.g. forth red dot, from left to right).

5. The MacKinlay and Pastor Enhancement is used to use the information on the covariance matrix of the residuals to make up for the missing factors that affect the mispricing of returns. This is achieved by generating a new covariance matrix of the form $\Sigma = \upsilon\mu\mu^T + \sigma^2 I_N$, where the parameters υ, μ and σ^2 are obtained from solving the likelihood function shown below [1]:

$$L(\mu, \upsilon, \sigma^2 | z_1, \dots, z_T) = |\mu\mu' \upsilon + \sigma^2 I|^{-\frac{T}{2}} \exp\{-\frac{1}{2}\sum_{t=1}^{T}(z_t - a)'(\mu\mu'\upsilon + \sigma^2 I)^{-1}(z_t - \mu)\}$$

The computed value for the parameters using the three securities used in exercise 4 (e.g. ASH, CRDA and SGE), were μ = $[4.045e^{-4}, 4.983e^{-4}, 1.186e^{-3}]$, $\upsilon = 6.963e^{-3}$ and $\sigma^2 = 3.01e^{-3}$. This resulted in a new covariance matrix:

$$\Sigma = \begin{bmatrix} 3.01e^{-4} & 1.26e^{-9} & 1.26e^{-9} \\ 1.26e^{-9} & 3.01e^{-4} & 1.26e^{-9} \\ 1.26e^{-9} & 1.26e^{-9} & 3.01e^{-4} \end{bmatrix}$$

This new covariance matrix was then used to plot the Markowitz efficient frontier, against the previous result. It can be observed that it offers a broader set of standard deviation options for efficient portfolios.

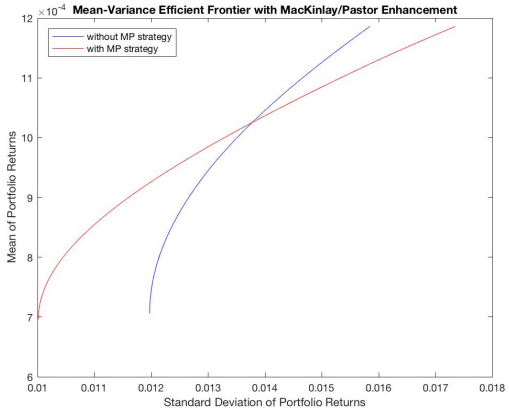

Figure 8: Comparison of Efficient Frontier with and without using MacKinlay and Pastor enhancement

6. a) A greedy forward selection algorithm was used to select the top 6 (e.g. a fifth) of the 30 constituent stocks from the FTSE 100, that best track the index. This was done by using Matlab's `sequentialfs` function as shown below.

```
train = @(x_train, y_train, x_test, y_test) sum((y_test - x_test*(x_train \ y_train)).^2);


counter = zeros(1,30);

for c = 1:50
    inmodel = sequentialfs(train,X,y,'cv',10);
    counter = counter + inmodel;
end

[B,I] = maxk(counter, 6);
```

Code 4: Greedy Forward Selection algorithm to select 6 stocks

The function `train` is the cost function used to provide a quantitative measure of how close a subset of selected features in matrix X is to the objective vector Y. This is computed using a simple linear regression cost function. The matrix X, of size 760x30, contains 760 sequential daily returns for each of the 30 available stocks and the vector Y, of size 760x1, contains the daily returns for the FTSE 100. The argument 'cv', represents the n-fold cross validation parameter, which in this case is 10. The algorithm was run for 50 times, each time keeping track of the chosen stocks, to plot a histogram depicting how many times each one was picked by the algorithm. This is shown below in Figure 9.

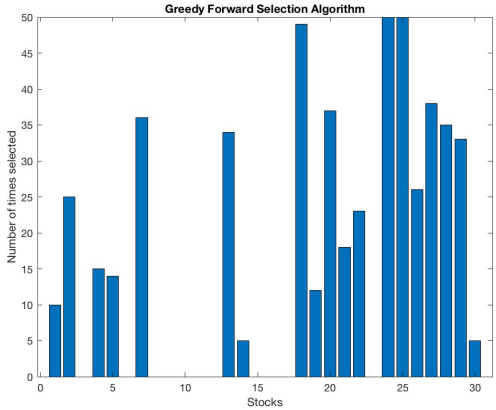

Figure 9: Stocks selected by Forward Selection algorithm

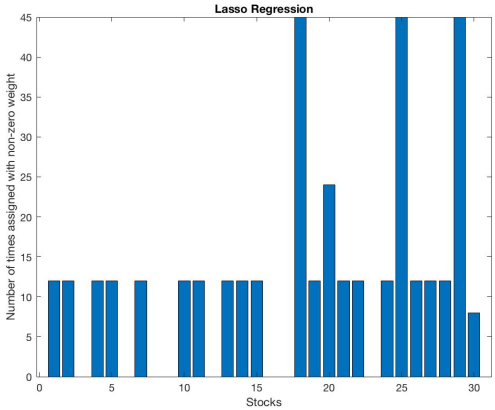

Figure 10: Stocks selected by Lasso Regression

The resulting 6 stocks were Stocks 7, 18, 20, 24, 25 and 29. To measure the correlation and similarity between this 6-stock portfolio and the FTSE 100, the Pearson's correlation coefficient and the expected return were computed. The correlation coefficient was calculated using MATLAB's `corrcoef` function, which returned a result of 0.1245. The expected return for this portfolio was $2.5142e^{-4}$, whereas the expected return for the FTSE 100 for the same time period was $1.2751e^{-4}$.

b) In this approach, a sparse index tracking portfolio was constructed by using l_1 regularization. Lasso regression was applied to the same X matrix and Y vector using the `lasso` function in MATLAB with a 10-fold cross validation. This was executed 50 times, and in each time, using the regularization parameter yielding the minimum MSE, the stocks for which the weights were assigned with a non-zero value were counted. The results are shown in Figure 10. For a visual presentation, Figure 11 illustrates the variation of the mean squared error (e.g. similarity to the FTSE 100) for different values of the regularization parameter. This was generated using MATLAB's `lassoplot` function.

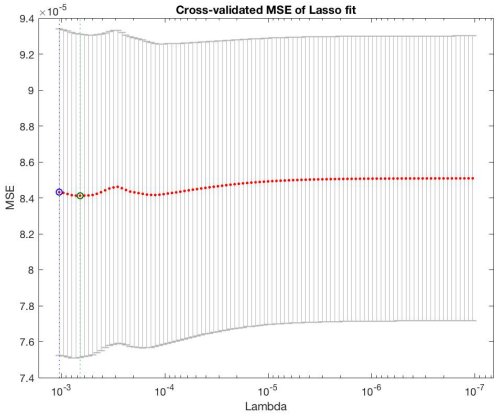

Figure 11: MSE against different values of the regularization parameter Lambda

Taking all this information into account, the regularization parameter which was chosen was 0.0005 which resulted in weights of 0.1416 for stocks 18 and 20, and 0.3584 for stocks 25 and 29, with the rest of the stocks set to 0. The Pearson correlation and the expected return obtained for this 4-stock portfolio were 0.155 and 1.2520e^{-4}, respectively. This shows that the Lasso regression resulted in a more accurate selection of stocks than the forward selection algorithm.

7. In the paper of Lobo *et al.*, transaction costs are introduced as functions $\phi(x)$ that take the transactions x, as inputs, and output the costs incurred from them. These are then included in the portfolio optimization portfolio problem as an additional constraint, leading to the following optimization problem:

$$\text{maximise } a^T(w+x) \qquad \text{subject to } 1^T x + \phi(x) \leq 0, \quad w + x \in S \tag{4}$$

where a^T is the expected returns of the assets, w are the current holdings and x the transactions, in which $x > 0$ when buying, and $x < 0$ for selling. The condition is valid for self-financing portfolios, as the condition only allows to buy securities by selling others to fill the cost. The second constraint simply encapsulates all the possible portfolios that can be built. The paper also suggest that optimization problem could be written in the following form:

$$\text{minimize } \phi(x) \qquad \text{subject to } a^T(w+x) \geq r_{min}, \quad w + x \in S$$

where the objective is to minimize the transaction costs subject to a minimum expected return r_{min}.

Figure 2 (from Lobo *et al.*'s paper) illustrates a cumulative distribution plot of the returns of a group of assets,100 risky and 1 riskless, in a portfolio. It maps the probabilities for a portfolio to generate a return of less than a value z, by assuming that they are normally distributed. The basis of the graph comes from the equation $\boldsymbol{Prob}(W \geq W^{low}) \geq \eta$, in which we want to calculate the probability that the holdings of a portfolio W, at the end of a particular timeframe, will be larger than W^{low}. This assumes that W is a Gaussian random variable, leading to the plot shown in Figure 2. Although there are some changes to the notation and representation of variables, the example given in Section 1.6 represents the same optimization problem as (4), with some tweaks and additions. The first addition is the short selling limit constraint given by $w_i + x_i^+ - x_i^- \geq s_i$, where the sold securities exceed the bough securities by more than s_i. The other additions are the of two shortfall risks constraints, with the two main parameters η and W_{low} (e.g. $\eta_1 = 80\%, W_{low}^1 = 0.9, \eta_2 = 97\%, W_{low}^2 = 0.7$) which assigns probabilities for the portfolio to produce a return

of W with $1-\eta$, as depicted in Figure 2 (from Lobo *et al.*'s paper). These could be added to our portfolio optimization problem, by first imposing the transaction costs constraints and defining these shortfall risks to much our risk admissibility.

(b)

1. The derivate of the cumulative normal distribution $N(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-\frac{x^2}{2}}$, is $N'(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$. To show that the expression (4) is true, we can work out d_1 and check if it matches its definition, as shown in (5).

$$SN'(d_1) = Ke^{(-r(T-t))}N'(d_2) \qquad (4) \qquad d_1 = \frac{\log(S/K) + (r + \partial^2/2)(T-t)}{\partial\sqrt{(T-t)}} \qquad (5)$$

First, we substitute $N'(d_1)$ and $N'(d_2)$ with their corresponding expressions to equation (4).

$$\frac{S}{\sqrt{2\pi}} e^{-\frac{{d_1}^2}{2}} = \frac{K}{\sqrt{2\pi}} e^{(-r(T-t))} e^{-\frac{{d_2}^2}{2}}$$

We then cancel out the constant $\sqrt{2\pi}$ and merge the exponentials together.

$$Se^{-\frac{{d_1}^2}{2}} = Ke^{(-r(T-t))-\frac{{d_2}^2}{2}}$$

We then move *K* to the left-hand side of the equation and apply *log* to both sides of the equation.

$$\log\left(\frac{S}{K}\right) - \frac{{d_1}^2}{2} = \left(-r(T-t)\right) - \frac{{d_2}^2}{2}$$

We then rearrange the equation to move everything but $\frac{{d_1}^2}{2}$ to the right-hand side of the equation and substitute d_2 in.

$$\frac{{d_1}^2}{2} = \log\left(\frac{S}{K}\right) + r(T-t) + \frac{{d_1}^2}{2} - \frac{d_1\partial\sqrt{(T-t)}}{2} + \frac{\partial^2(T-t)}{2}$$

We cancel out $\frac{{d_1}^2}{2}$, and work out for d_1.

Ans. a)
$$d_1 = \frac{\log\left(\frac{S}{K}\right) + r(T-t) + \frac{\partial^2}{2}(T-t)}{\partial\sqrt{(T-t)}} = \frac{\log\left(\frac{S}{K}\right) + (r + \frac{\partial^2}{2})(T-t)}{\partial\sqrt{(T-t)}} = (5)$$

The derivative for $\frac{\partial d_1}{\partial S}$ can be calculated by first isolating the term that contains S, as shown below.

$$d_1 = \frac{\log(S)}{\partial\sqrt{(T-t)}} - \frac{\log(K)}{\partial\sqrt{(T-t)}} + \frac{(r + \frac{\partial^2}{2})(T-t)}{\partial\sqrt{(T-t)}}$$

And then applying the derivative.

$$\frac{\partial d_1}{\partial S} = \frac{1}{S\partial\sqrt{(T-t)}}$$

Because $d_2 = d_1 + constant$,

Ans. b)
$$\frac{\partial d_1}{\partial S} = \frac{\partial d_2}{\partial S} = \frac{1}{S\partial\sqrt{(T-t)}}$$

To compute $\frac{\partial c}{\partial t}$, we start by applying the derivative product rule to the second term of the equation.

$$\frac{\partial c}{\partial t} = SN'(d_1)\frac{\partial d_1}{\partial t} - Ke^{(-r(T-t)}N'(d_2)\frac{\partial d_2}{\partial t} - rKe^{(-r(T-t)}N(d_2)$$

We can then substitute equation (4) to replace the second term of the equation with $SN'(d_1)$.

$$\frac{\partial c}{\partial t} = SN'(d_1)\frac{\partial d_1}{\partial t} - SN'(d_1)\frac{\partial d_2}{\partial t} - rKe^{(-r(T-t)}N(d_2) = SN'(d_1)\left(\frac{\partial(d_1 - d_2)}{\partial t}\right) - rKe^{(-r(T-t)}N(d_2)$$

Because we know that $d_1 - d_2 = \partial\sqrt{(T-t)}$, we can compute $\frac{\partial(d_1-d_2)}{\partial t}$, by applying the chain rule using a temporary variable $u = T - t$ and performing $\frac{\partial}{\partial t} = \frac{\partial}{\partial u} * \frac{\partial u}{\partial t}$.

$$\frac{\partial}{\partial u} = \frac{\partial}{2}u^{-1/2} \qquad \frac{\partial u}{\partial t} = -1 \qquad \frac{\partial}{\partial t} = -\frac{\partial}{2\sqrt{(T-t)}}$$

Finally, we substitute this result into $\frac{\partial(d_1-d_2)}{\partial t}$ and we arrive to the final equation.

Ans. c) $$\frac{\partial c}{\partial t} = -rKe^{(-r(T-t)}N(d_2) - SN'(d_1)\frac{\partial}{2\sqrt{(T-t)}}$$

The partial derivative $\frac{\partial c}{\partial S}$ can be calculated by first applying the product rule to the first term.

$$\frac{\partial c}{\partial S} = N(d_1) + S\frac{\partial N(d_1)}{\partial S} - Ke^{(-r(T-t)}\frac{\partial N(d_2)}{\partial S}$$

We can then use the chain rule $\frac{\partial N(d)}{\partial S} = \frac{\partial N(d)}{\partial d} * \frac{\partial N(d)}{\partial S}$ to write it in the following form.

$$\frac{\partial c}{\partial S} = N(d_1) + SN'(d_1)\frac{\partial d_1}{\partial S} - Ke^{(-r(T-t)}N'(d_2)\frac{\partial d_2}{\partial S}$$

By substituting the equations (4) and Ans. (b), the second and third terms are equivalent and therefore, they cancel each other out. Leading to.

Ans. d) $$\frac{\partial c}{\partial S} = N(d_1)$$

To calculate the second derivative $\frac{\partial^2 c}{\partial S^2}$, we apply another derivative to Ans. (d), which leads to the previous chain rule $\frac{\partial N(d)}{\partial S}$ =We then simply substitute the equation from Ans. (b) to reach to the expression below.

$$\frac{\partial^2 c}{\partial S^2} = \frac{\partial N(d_1)}{\partial S} \qquad \frac{\partial^2 c}{\partial S^2} = \frac{\partial N(d_1)}{\partial d} * \frac{\partial N(d_1)}{\partial S}$$

Ans. e) $$\frac{\partial^2 c}{\partial S^2} = N'(d_1)\frac{1}{S\partial\sqrt{(T-t)}}$$

Finally, we can prove that the expression for the call option price in indeed the solution to the Black-Scholes differential equation by substituting Ans. c), d) and e) in the equation below.

$$\frac{\partial c}{\partial t} + \frac{1}{2}\partial^2 S^2\frac{\partial^2 c}{\partial S^2} + rS\frac{\partial c}{\partial S} - rc = 0$$

Where all the terms cancel out as expected.

2. MATLAB's `blsprice` function was used to compute the Black-Scholes Model for each of the option prices. The arguments used to achieve this were the *strike price*, given in the file name; the *price* of the security at a given day; the *annual interest rate*, set to 6.0%; the *time*, given in a yearly scale; and the Hull *volatility*, which was computed using the equation $\partial\sqrt{T}$, where ∂ is the standard deviation of the proportional price change and T, the timescale computed in total number of trading days in a year, or 252. The Hull volatility was calculated each using the precedent 52 past prices of the security. Figures

12 and 13 illustrate a comparison between the actual option prices given in the dataset, against the option prices computed using the Black-Scholes Model.

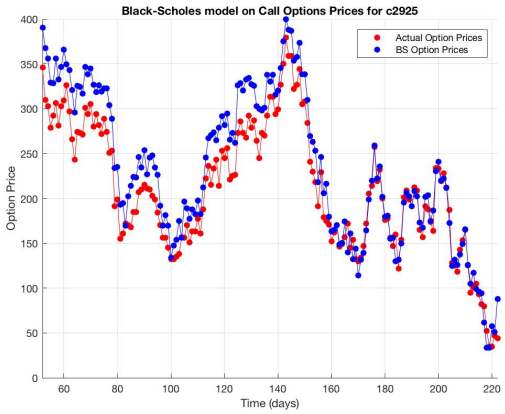

Figure 12: Actual Option Prices vs BS Option Prices for c2995

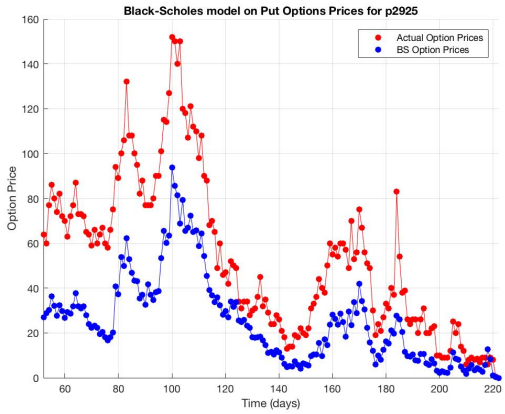

Figure 13: Actual Option Prices vs BS Option Prices for p2995

It was consistently observed that for the call options generated by the Black-Scholes Model, the values were slightly higher to those of the actual prices, whereas for the put options, the values were lower. To quantify the differences between actual prices and the BS model, a histogram of the MSEs was plotted for all the different strike options, as shown in Figure 14. The bar plot shows that as the strike price of the call options increases, the Black-Scholes model generates values that are closer to the actual prices, whereas for the put options, it's the opposite. The implied volatility was computed for the call option, at a strike price of 2925, for a random 30-day period, resulting in the scatter plot shown in Figure 15. This shows that the implied volatilities from days 100 to 129, have a much higher variation than the Hull volatilities, previously computed.

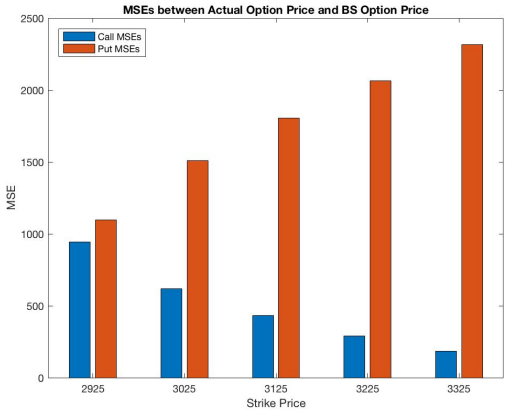

Figure 14: MSE for different strike values

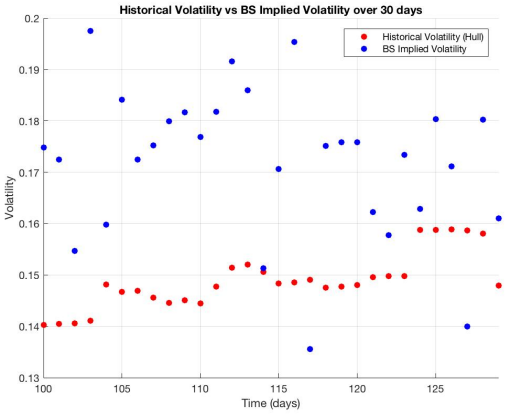

Figure 15: Hull Volatility vs Implied Volatility (Days 100-129)

Lastly, the implied volatilities for day 100 were calculated and plotted for each strike price as it can be seen below in Figure 16. The plot shows that there is an upward slope for the put option implied volatilities and the opposite for the call options. This is property is known as the *volatility smile*.

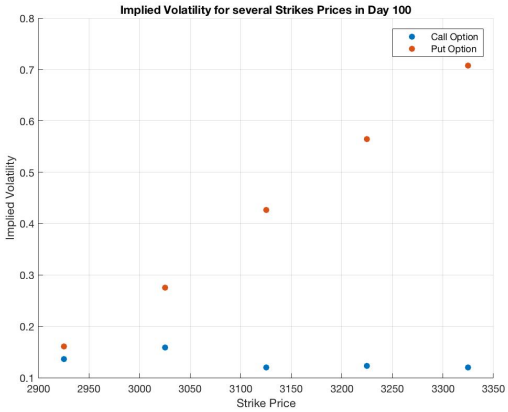

Figure 16: MSE against different values of the regularization parameter Lambda

3. b) A simulation using MATLAB's `newrb` function was performed to verify the claims from the paper by Hutchison *et al.*, where they suggest that the complex parametric relationship between the option prices and their variables can be approximated by 'learning' their respective Black-Scholes model using radial basis functions. The network structure used to train these RBF networks consisted of 2 layers and 150 neurons. Figure 16 shows how the MSE changes as the number of neurons in the network is increased.

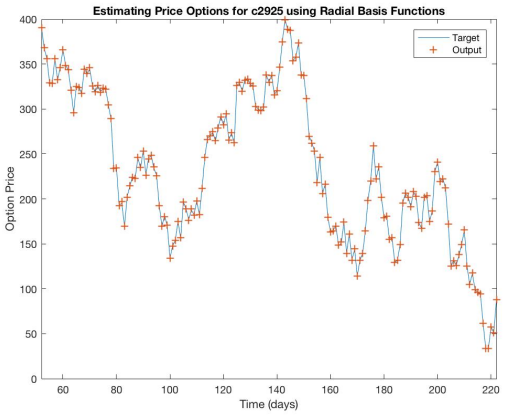

Figure 17: Black-Scholes Option Prices against RBF for c2995

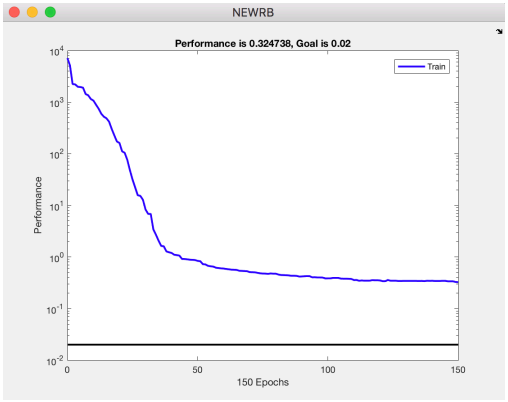

Figure 18: MSE for each epoch (number of neurons) for c2995

The results shown in Figure 16 and 19, show that the RBF network almost perfectly mirrors the Black-Scholes options prices, proving that the claims made by Hutchison *et al.* are indeed valid. The MSEs for each of the strike prices was also plotted as a bar plot, as shown in Figure 18, which demonstrates that the MSEs are very small.

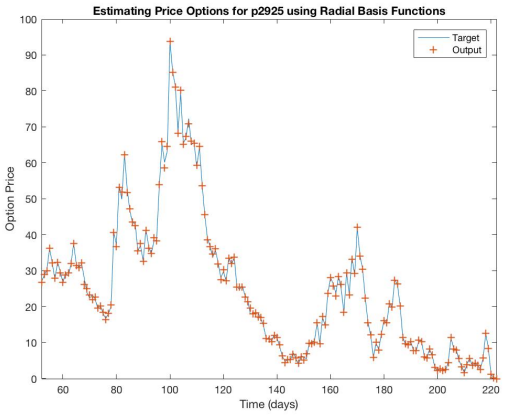

Figure 19: Black-Scholes Option Prices against RBF for p2995

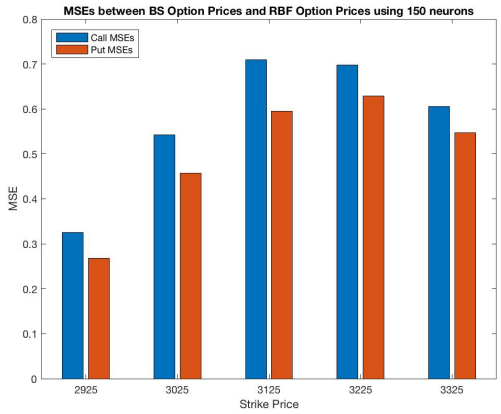

Figure 20: MSEs between BS and RBF option prices

Furthermore, the same RBF network setup was used to compute the *Delta* ($\Delta = \partial C/\partial S$) for the option prices. The results are illustrated in Figure 21. Again, the RBF network computed delta values which are very close to those obtained from the Black-Scholes model using the MATLAB function `blsdelta`, with a MSE of 0.007.

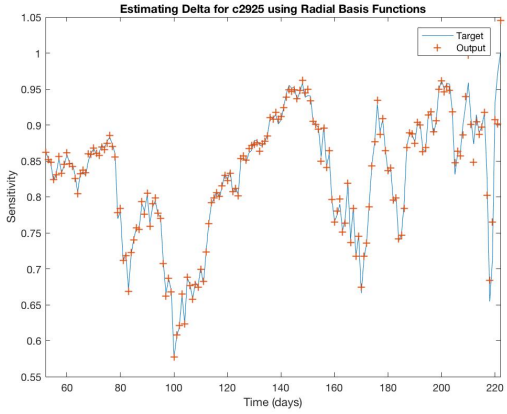

Figure 21: Δ predictions using a RBF network

Tickers of the 30 FTSE 100 constituent stocks used for the exercises

ATH – ADM – SGE – AAL – ANTO – BATS – III – FERG –ABF – CNA – CCH – CRDA – RIO – CRH – CPG – IHG – ITRK – HSBA – IMB – INF – PPB – PSON – NXT – NMC – OML – SHPG – SKY – SMT – SGRO – SVT - FTSE

Source: uk.finance.yahoo.com

References

[1] *A. C. MacKinlay, L. Pastor, "Asset Pricing Models: Implications for Expected Returns and Portfolio Selection", NBER Working Paper No. 7162, pp. 6, 1999.*

Code

https://github.com/leedanieluk/finance_assignment

Name: Daniel Lee
Student ID: 2589549**4**

12/03/18