

**Group Coursework Submission Form**

Specialist Masters Programme

|  |  |  |
| --- | --- | --- |
| **Please list all names of group members:**  (Surname, first name)  1. Cutrera, Vittorio  2. Daubresse, Alexia  3. Fayed, Ahmed | 4. Lee, Jeonghae  5.  6.  **10**  7.  **GROUP NUMBER:** | |
| **MSc in:**  Energy, Trade and Finance/Finance | | |
| **Module Code:**  SMM238 | | |
| **Module Title: VBA with Applications for Finance** | | |
| **Lecturer: Oliver Manyemba** | | **Submission Date: 20.07.2020** |
| **Declaration:**  By submitting this work, we declare that this work is entirely our own except those parts duly identified and referenced in my submission. It complies with any specified word limits and the requirements and regulations detailed in the coursework instructions and any other relevant programme and module documentation. In submitting this work we acknowledge that we have read and understood the regulations and code regarding academic misconduct, including that relating to plagiarism, as specified in the Programme Handbook. We also acknowledge that this work will be subject to a variety of checks for academic misconduct.  We acknowledge that work submitted late without a granted extension will be subject to penalties, as outlined in the Programme Handbook. Penalties will be applied for a maximum of five days lateness, after which a mark of zero will be awarded. | | |
| **Marker’s Comments (if not being marked on-line):** | | |

**Deduction for Late Submission: Final Mark:**

**%**

# Executive summary

While VBA and Excel are similar in many ways, VBA represents a big advantage by automating procedures and therefore making financial models and analysis much faster and efficient. VBA automates usual Excel procedures enabling models to run faster, for instance by using shortcut keys.

The herein report aims to discuss the advantages of adopting a VBA approach instead of a normal Excel sheet to a range of financial tasks varying from portfolio management to saving accounts and option pricing. The paper is structured as follows: first, we discuss Question 3, which uses Monte-Carlo simulation to compute stock prices, then we move on to Question 4, and the problem of portfolio optimization, before explaining the code of Question 7 concerned with option pricing, and finally, Question 8, related to retirement planning.

For each question, we outline the differences between a standard (Excel) and a VBA approach, paying particular attention to the increased easiness in modifying *Sub* procedure and functions for the final output. Furthermore, enhanced functionalities are explained with respect to of each code and separate code parts are analyzed in order to explain the backbone of the VBA functionalities to the user.

# Table of Contents

[Executive summary 1](#_Toc45701059)

[Question 3 1](#_Toc45701060)

[Introduction 1](#_Toc45701061)

[Code 1](#_Toc45701062)

[Question 4 3](#_Toc45701063)

[Overall structure 3](#_Toc45701064)

[Improved functionality 4](#_Toc45701065)

[Code 4](#_Toc45701066)

[Testing 5](#_Toc45701067)

[Question 7 5](#_Toc45701068)

[Declaring Variables 5](#_Toc45701069)

[Developing Binomial Option Tree 6](#_Toc45701070)

[Binomial Option Pricing Tree Calculations 7](#_Toc45701071)

[Conclusion 7](#_Toc45701072)

[Question 8 8](#_Toc45701073)

[VBA advantages 8](#_Toc45701074)

[Conclusion and recommendations 8](#_Toc45701075)

# Question 3

## Introduction

Assume that you invest in equity. You will wonder how much profit you will make and what would be your losses in case of a downturn.

The purpose of this model is to visualize the price path of a stock using Monte Carlo simulation. Monte Carlo simulation refers to a methodology that solves a problem by generating random situations with random variables. The price of the equity used in this model is based on the Brownian motion.

## Code

Equation (1) finds *S(T),* the stock price after period *T*, in this case one year later, or 250 business days. *S(0)* is the current stock price, *r* is expected return, *σ* is the volatility, and *Z* is the standard normal random variable with mean 0 and variance 1.

(1)

With this model, investors can presume the future stock price based on the given current price, expected return, volatility and the simulation step size. It will show the price path from now to one year later. Users can easily get the current price of the stock. By using historical data such as ROE (return on equity), users can calculate the expected return, and can obtain volatility (based on the standard deviation). Users can get other variables from the market data, but they have to choose themselves how many times they will simulate the model. Thus, if the user input the simulation step size 10, it will operate the simulation 10 times.

We used the *For/Next* loop to build the 10 steps simulation. We used simulation step size variable *t* and outer loop in the code to take a simulation step size that the users write.

Each simulation runs 250 times the calculation in the inner loop. In the first iteration, the price *S(T)* would be the current price so we used the *If/Then* function to use current price in the first iteration in the inner loop. From the second iteration, the loop calculates two factors. First, it calculates *Z-value*.

*[1] rdraw = Application.NormSInv(rnd)*

Using the above code, we are able to draw a random variable from the inverse of cumulative standardized normal distribution. Within the inner loop, this statement inputs the random number between 0 and 1 in the *rdraw* variable which can be used as the *Z*-*value*. To show the expected price in each day, we used the below code:

*[2] Cells(28 + i, 2 + m) = Cells(27 + i, 2 + m) \* Exp((r - 0.5 \* (v ^ 2)) \* (1 / 250) + v \**

*\_ Sqr(1 / 250) \* rdraw)*

We converted the Brownian motion statement to VBA code. This code estimates the price of day ‘*t’* using the price of the day ‘*t-1’*. The time increment of the code is one day, so we used 1/250 of period *T* in above Brownian motion (1). The old price which is the price of day *‘t-1’* is saved in the *(27 + i, 2 + m)* cell. The new price will be saved in the *(28 + i, 2 + m)* cell which is one cell below the previous price. After operating the inner loop for 250 times, it will input new simulation results in the next column, if the user’s step simulation variable is larger than 1.

To make the model, we set default value for the variables that we mentioned above but give users the ability to change the variables on the spreadsheet. When a user clicks the ‘*simulation’* button, the model will operate simulations according to the entered step size and will show the results for each simulation. The user can see the figures changing according to the dates. Moreover, we draw a line chart to help users estimate and visualize price changes:

*[3] Worksheets(“Question3”).Shapes.AddChart.Select*

*[4] ActiveChart.SetSourceData Source:=Range("B28").CurrentRegion*

In code [3] *Addchart* function is used to draw the chart in the “Question3” sheet (GreenStephen, 2011). Drawing the chart, it is important to choose the source. In code [4], we choose the region that the cell (*B28*) belong to. We chose the cell (*B28*) because it shows the price of the first simulation on the first day, so it can be recognized as the region regardless of what simulation step size the user has chosen, if it is not zero.

The model will generate the chart and data table every time the user clicks the ‘*simulation’* button. We could insert the chart in the ‘*insert’* tab in the Excel spreadsheet, but we decided to generate chart with VBA because the model can cope with various situation when we use VBA. For example, if we used a ready-made-chart with the maximum band 20 and minimum band 10, when the user inputs a price larger than 20, the chart doesn’t show the price path properly. In addition, if we used a ready-made-chart which has the already-selected region, it cannot select the region properly after the simulation step size has changed. This method has the disadvantage of taking more time operating the model, but it is user friendly as it gives more choice to users and they don’t need to select the region every time when operating the model.

As the chart and data table are generated separately every time the user operates the model, the exiting chart and data table have to be deleted. To execute this we used the below codes:

*[6] Rows(27 & ":" & 278).Select*

*[7] Selection.Clear*

*[8] Worksheets("Question3").ChartObjects(1).Delete*

In code[6], it selects the rows from 27 to 278 where the data table belong, and delete the data in the selected rows. In code[8], it deletes the existing chart in the spread sheet named “Question3”.

Users can use this model before buying the equities, presuming the price after a year and the best or the worst scenario that users can encounter during the investment.

# Question 4

## 

## Overall structure

The aim of the herein code is that of giving the user control over his investment portfolio.

The investor will select its share of wealth to invest namely, the portfolio size, as well as category and asset class.

The user has control over the portion of monies to allocate to each category; the model of this particular code includes: commodities, stock indices and bonds. The portion to the previous categories will be accessible through the worksheet ‘*inputs*.

The commodity category is composed by four indices structured on rolling futures contracts, each composed by a different commodity: agricultures, copper, platinum and palladium. The stock indices category gives choice over three sectors of the economy: healthcare devices, consumer staples and utilities. Finally, the bonds category provides the user with two bond indices: one composed of treasury bills of intermediate maturity and the other by corporate fixed income.

The investor has also the freedom to allocate different portions of its wealth to every asset class described above, specifically in the worksheet ‘funds’.

Finally, the code will run producing the output on the ‘portfolio’ worksheet building an investment portfolio according to the investor choices.

## Improved functionality

This model performs best when used for structuring diverse portfolios. As the user will have more funds available to invest, he will be able to simply add them to the initial input of the portfolio size, the same is also true for each category as well as asset class.

When compared to similar models programmed with Excel instead of VBA, the latter has the advantage of not showing the category or fund where monies were not allocated to the portfolio making output sheet clearer.

Furthermore, an Excel built model would most probably have all the inputs in a single worksheet to avoid changing it every time when building the portfolio, however, this will be confusing for the final user. On top of that when indeed the total funds are allocated to only few asset classes the output will result visually unstructured and dispersed.

## Code

The variables defined in the first part are needed to guide the entire flow of the coding throughout its sections. It is important in this section to clearly and concisely define a variable in order to be easily understandable from the user as well as easily typeable as so avoiding typo errors when repeating it in other sections.

Following are given instruction on how structuring the interface in the three different worksheets and the position of the inputs on the excel grid.

When certain inputs are to be used, the worksheet in which they are must be activated, later it is possible to proceed with the denomination and form of each variable to be used in that worksheet. Specifically, in the Read Inputs section, the percentages as of the portfolio value for each investment category are set as well as their value in monetary terms. For both percentage and dollar terms of the Bonds category a deductive approach has been adopted in attributing the percentage as to complete 100%. When the portfolio sheet is activated specific cells are chosen for the output of related values as well as their precise format such as number of decimals, font and size.

We take the example of the commodity category to explain how a bridge is built between two worksheets and how to find a determined keyword, in a selected range of cells:

*typeR = Application.Match("Commodity", Range("A1:A300"), 0).*

The if statement was chosen as method to condition the execution of the program flow.

In our specific case to find the keyword, in this case always commodity and its related value, indicating which worksheet to copy from and the one for its output:

*Sheets("Funds").Range("A" & r1 & ":B" & r1).Copy \_*

*Sheets("Portfolio").Range("A" & rOut)*

The if conditions can be modelled with ‘else, true or false’ in order to stop them or go to another function, in the herein case:

*If Cells(r1, 1).Value = "" Then Exit For’ was used to end the loop and later on closed ‘End If, Next r1’.*

In order to calculate, in this example the stock indices funds have been used another variations of the if flow:

*If Sheets("Inputs").Range("A" & r2).Value <> "" And \_*

*Sheets("Inputs").Range("C" & r2).Value <> 0*

Then so as testing two statements and moving on to values and position determination.

Lastly, in the bottom part of the code we can find the instructions given by the code for different formatting of rows and columns alignment as well as font and sizes, this time the method employed was that of sub procedures. Its peculiarity is that of carrying out tasks and manipulating objects.

## Testing

It is really handy the verification that all the funds were allocated correctly since in

the “Portfolio” sheet is shown the portfolio value, that will correspond to the initial input.

Subsequently the user will easily change its inputs from both percentage and dollar allocation to the categories and funds being able to immediately see any wrongdoing given the fact that the final result would not correspond to what was originally planned as total investment.

# Question 7

## Declaring Variables

The below code declares two separate arrays for stock prices and an array to store the time in each step. Other variables were declared and were assigned a value on the spreadsheet, therefore the cells changed to respond to the input data. *Div* and *exdiv* are input variables.

A screenshot of a cell phone

Description automatically generated

|  |  |
| --- | --- |
| Variable | Description |
| *dt* | The steps that will occur in years. |
| *u* | If the option goes up it will use the u multiplier. |
| *d* | If the option goes down it will use the d multiplier |
| *emrdt* | Discount factor to give the present value of option. |
| *Su* | Calculates the uncertain component. |
| *P* | The probability the tree will take an up (u) movement. |

## Developing The Binomial Option Tree

A screenshot of a cell phone

Description automatically generated

The code below describes the basis of the binomial option tree (the last line indicates that the code starts from time 0):

## Binomial Option Pricing Tree Calculations

A screenshot of a social media post

Description automatically generated

The *For* loops are used to create the stock price tree. Calculations estimate at what time each step occurs and the present value of the dividends at this time. After this, the code calculates the values of the option for the tree of the uncertain part of the stock price, before calculating the stock price. The four loops generate the option value tree starting from maturity, by using the Excel *max* function. The earlier nodes are calculated as present values of the expected values, both for American and European models. Then the code calculates the option value from the tree at time 0. Finally, the American Option Model computes the immediate exercise payoff; this is where the higher values are chosen at each node to decide whether the option should be exercised early or not.

With a large number of steps the model can produce accurate estimates for the option value. The model can be changed to allow for other known dividends.

# Question 8

## VBA advantages

To answer question 8, we used ‘Sub’, a procedure, which contains most of VBA programs and is similar to a user-defined function (Module handbook 2). We used a VBA model which included Solver to optimise retirement plan calculations. Our approach and the order of procedures closely followed the ones which were used in class.

## Code

In the model, we used *Option* *Explicit* in order to specifically declare certain variables as *Dim*. Using this option, those items which were not explicitly declared as *Dim* are automatically treated as Variant type. This option also helps to avoid typing the name of an existing variable incorrectly. The advantage of using *Dim* at the module level is that *Dim* variables become available to all procedures within that module. The table below summarises and explains the variables defined in the code:

|  |  |
| --- | --- |
| Variable | Description |
| *initBal* | Initial balance of savings |
| *pvEndBal* | The desired end balance of savings in today’s dollars |
| *currAge* | Current age of the saver |
| *retAge* | Retirement age of the saver |
| *nYrs* | Number of years till retirement |
| *taxBasis* | The income tax basis |

After declaring variables and assigning the given values to cells in column 2, we wrote in VBA the equations needed to calculate the Year Beginning Balance, Annual Savings Payment, and the Savings Balance at the end of the year separately for year 1, because their formulas changed in consecutive years. For the rest, we defined the same equations for all the years and ordered VBA to fill in the table of our chosen range.

Furthermore, we defined functions for the target savings balance in today’s dollars (*pvEndBal*) and the Final after-tax balance less target balance (*Cell(20,2)*), which we then used to run *Goal Seek.*

For this question, another alternative to a VBA model would be to input individual values in the cells, define functions for certain cells (for instance, *Year Beginning Saving Balance* for years 2 to 23), run *Goal Seek* to find the desired future value of the *Savings Balance*, and from there find the required savings amount for year 1. Using VBA, however, we are able to run a program based on entered values and incorporate *Goal Seek* inside the code, so it is executed automatically when the macro is launched.

*' Run Goal Seek*

*Range("B20").GoalSeek Goal:=0, ChangingCell:=Range("B17")*

When creating the model, one problem which came up when we were trying the model for the first time was linked to successfully running *Goal Seek*. This was due to cell names which were not matching, and because Excel had issues renaming the relevant cells. However, it was then resolved by copying the module to a new worksheet.

# Conclusion and recommendations

In this report, we outlined the differences between Excel and VBA and proposed VBA models to solve the given problems, explaining along the way the advantages or disadvantages of using VBA over Excel.

At first, learning and writing a code may be longer than filling a simple Excel sheet, however in the long term, using a VBA model can be applied to different dataset and kept in Excel for future usage. Some differences in procedure include the use of loops, where the same variable is used to calculate outcomes under a certain condition: VBA wouldn’t store intermediate variables in its calculations. Moreover, VBA has to be given exact instructions for the order of the sequence to successfully implement the model. Thus, we’d have to make all values used in VBA readily available for calculations.

Perhaps some people may find Excel more convenient because there is no need of checking if all the variables used in calculations are correct and it is easy to overwrite variables to rectify calculations (Module handbook 2).

VBA is critical in order to optimize Excel functions, and if programmed efficiently, may provide real time updates and a much user-friendly interface. Several errors can be avoided and detected immediately but most importantly modifying the function through the code is much quicker.

VBA is a language that everyone in the financial industry should know due to the sort of operations are carried which can be enhanced with this programming language.

References

Glasserman, P. (2013). Monte Carlo Methods in Financial Engineering. In P. Glasserman, *Monte Carlo Methods in Financial Engineering* (p. 596).

John Green, S. B. (2011). Excel 2007 VBA Programmer's Reference. In S. B. John Green, *Excel 2007 VBA Programmer's Reference* (p. 1176).

Manyemba, O. (2020) Module Handbook 2