Ben Coburn

IS 520

Dr. Allen

12/10/15

Executive Summary

Active stock traders often find themselves in need of high-frequency trading algorithms that are tailored to the behavior of specific stocks. In order to meet this need, the VBA-based solution presented here allows users to "test-drive" custom algorithms on any publicly traded stock, bond, or index fund. The program has two parts: (1) Refresh Stock Data, and (2) Algorithm Simulator.

Refresh Stock Data. First, the program allows users to input a ticker symbol in the appropriate cell and query the web for the most recent two weeks of historical stock data. This is accomplished via the "Refresh Stock Data" button on the Algorithm ribbon. Historical data is queried and parsed in the background, then presented for the user on the "Data" worksheet. The following data fields are presented:

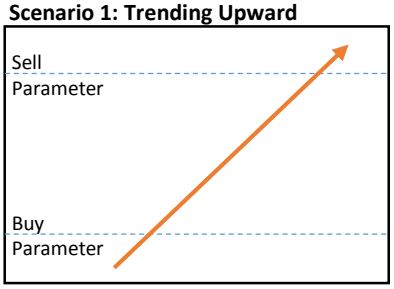
- Stock price (minute-by-minute granularity)
- Minute of the day (e.g., 7:30:00 AM)
- Relative trading day (e.g., -2 represents data from two days ago)
- Daily percent change (the percent change from the opening price on that specific trading day)

Algorithm Simulator. The second part of the program allows the user to define and test an algorithm on the two weeks of historical data. This functionality is accessed through the "Algorithm Simulator" button on the Algorithm ribbon. The simulator first presents a user form, prompting the input into three fields:

- Principle (i.e., how much do you want to simulate investing?)
- Buy Parameter (the algorithm will invest the principle when the stock has risen by a certain percentage)
- Sell Parameter (the algorithm will sell its shares when the stock has risen beyond the buy parameter to a certain percentage)

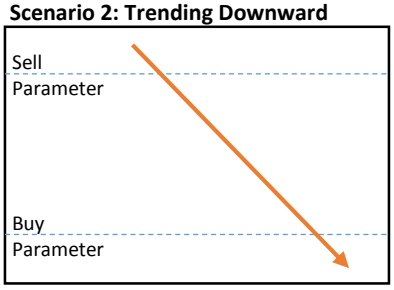
The program then runs the algorithm on the historical data. The algorithm is a momentum based algorithm that functions according to the following rules:

Scenario 1: Trending Upward



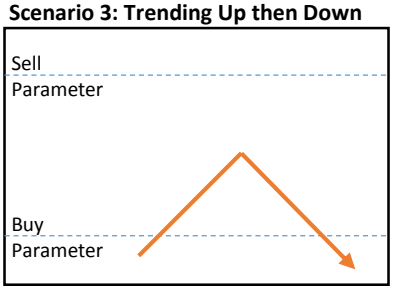
Outcome: Algorithm buys shares at the "buy parameter" and sells shares at the "sell parameter"

Scenario 2: Trending Downward



Outcome: Algorithm does not buy any shares when crossing the "buy parameter from above"

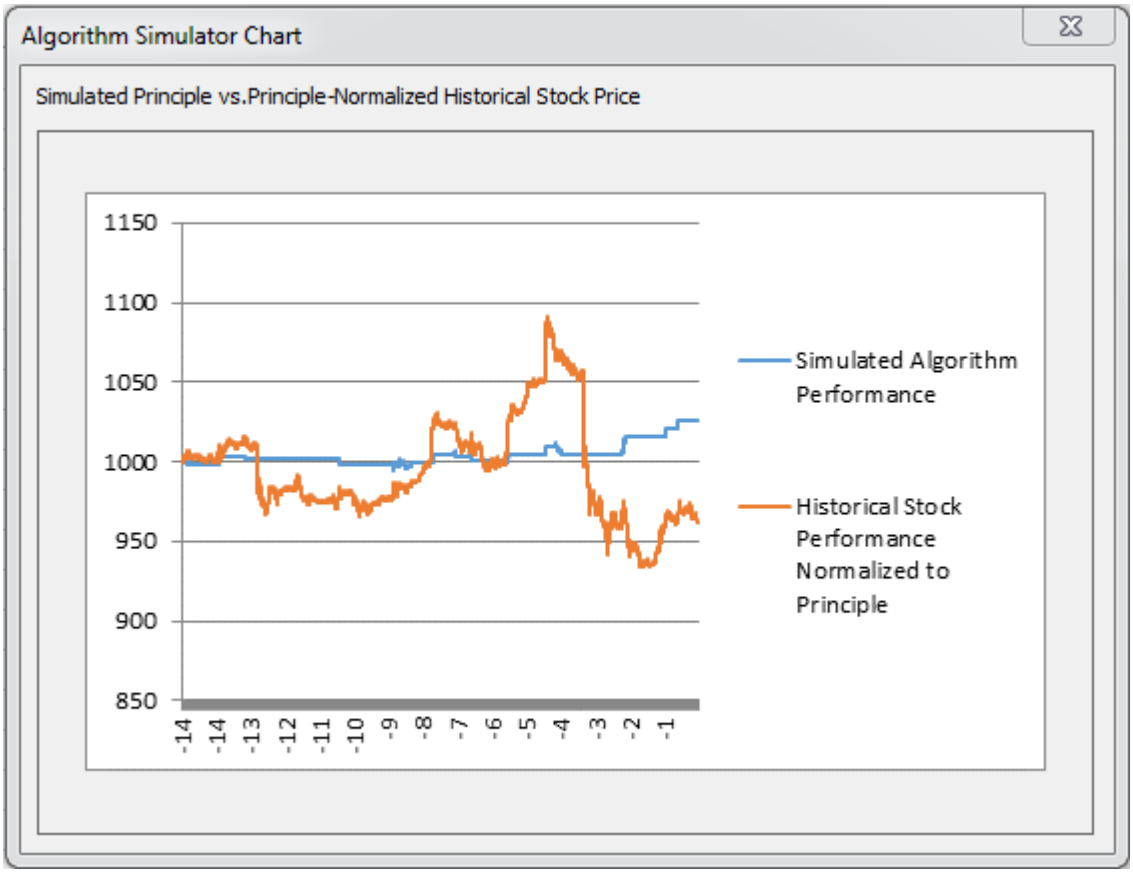
Scenario 3: Trending Up then Down



Outcome: Algorithm buys shares at the "buy parameter" and again sells shares at the "buy parameter" to minimize losses

Immediately after the algorithm has run, the user is presented with a message box detailing the performance of their algorithm in absolute dollar terms and percentage terms. This allows the user to see what their returns *would have been* had they been running their algorithm on the actual market.

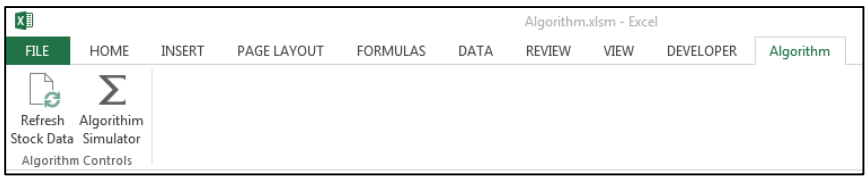
Lastly, the user is presented with a user form showing a graph of their algorithm-simulated principle versus the stock's historical performance (normalized to the amount of the original principle). For example:



The user is then free to re-run the algorithm again with updated parameters in hopes of finding an algorithm even better suited to the specific stock (or a new stock). Each time that the user runs a specific algorithm, the inputted values become the default values the next time the simulator is run.

Implementation Documentation

This VBA-solution uses one module and two user forms. The module is primarily used to refresh stock data, while the user forms are primarily used for the algorithm simulator. All arrays in this solution are specified to Option Base 1. All procedures in this solution are initially activated via buttons on the customer "Algorithm" ribbon:



Module 1. This module is primarily used to refresh stock data. It contains three subroutines: (1) Algorithm, (2) refreshData, and (3) parsePasteData.

1. The Algorithm sub simply turns off screen updating, calls the next two subs, and turns screen updating back on. "Algorithm" is initiated by the user clicking "Refresh Stock Data" in the custom ribbon:

```
Sub Algorithm(button As Object)

Application.ScreenUpdating = False

refreshData
parsePasteData

Application.ScreenUpdating = True

End Sub
```

2. The refreshData sub un-hides the sheet with the web query, and runs the web query on a Google Finance CSV webpage using a custom URL concatenated to include the stock ticker. If the ticker does not return any values, the code recognizes that the ticker is invalid and notifies the user via a message box. The sub the hides the web query sheet:

```
Sub refreshData()

'This procedure refreshes the query of historical stock data on a hidden tab
getDataSheet.Visible = True
getDataSheet.Activate

With getDataSheet.Range("A3").QueryTable
 .Connection = _
 "URL;http://www.google.com/finance/getprices?i=60&p=300d&f=d,o,h,l,c,v&df=cpct&q=" & Sheet2.Range("B1")
 .WebSelectionType = xlAllTables
 .WebFormatting = xlWebFormattingNone
 .WebPreFormattedTextToColumns = True
 .WebConsecutiveDelimitersAsOne = True
 .WebSingleBlockTextImport = False
 .WebDisableDateRecognition = False
 .WebDisableRedirections = False
 .Refresh BackgroundQuery:=False
End With

getDataSheet.Visible = False

'Gives error message if invalid ticker, ends all subs
If getDataSheet.Range("A10") = "" Then
 MsgBox "You have entered an invalid ticker symbol"
 End
End If

End Sub
```

3. The parsePasteData sub parses the queried data off of the hidden query sheet and pastes it in an intuitive format on the main worksheet ("Data"). The original CSV format of the queried data is as follows:

	A
1	Ticker:
2	
3	EXCHANGE%3DNYSE
4	MARKET_OPEN_MINUTE=570
5	MARKET_CLOSE_MINUTE=960
6	INTERVAL=60
7	COLUMNS=DATE,CLOSE,HIGH,LOW,OPEN,VOLUME
8	DATA=
9	TIMEZONE_OFFSET=-300
10	a1448029800,119.11,119.11,119.11,119.11,400
11	1,119.2699,119.27,119,119.01,557967
12	2,119.1818,119.27,119.11,119.2,21409
13	3,118.984,119.22,118.984,119.12,28414
14	4,119.03,119.07,118.9,118.99,20375
15	5,119.1836,119.2399,119,119.04,18260
16	6,119.3,119.35,119.13,119.16,32244
17	7,119.34,119.3525,119.23,119.32,17340
18	8,119.37,119.49,119.3101,119.324,19684
19	9,119.4595,119.47,119.36,119.36,24307
20	10,119.43,119.5,119.35,119.4494,49374

Row 7 of the query explains what each of the comma separated values in rows 10+ correspond to.

Etc.

The parsePasteData sub dimensions an array for each of the four key variables: stock price (priceArray), minute of the day (timeArray), relative trading day (dayArray), and daily percent change (changeArray). After counting the number of rows of data for the most recent stock query, the sub re-dimensions each array to match the number of rows. This is accomplished with the following code:

```
Sub parsePasteData()

'This procedure parses the historical stock data to get stock price, day, and time
'It then pastes it to the appropriate location
Dim dayArray() As Integer
Dim priceArray() As Double
Dim timeArray() As Date
Dim changeArray() As Single
Dim x As Single

'Sets all arrays equal to the amount of "observations" for available for that stock
x = WorksheetFunction.CountA(Worksheets("getDataSheet").Range("A1:A10000")) - 9
 ReDim priceArray(x)
 ReDim dayArray(x)
 ReDim timeArray(x)
 ReDim changeArray(x)
```

Next, the sub uses a For-Next loop to assign values to priceArray, timeArray, and dayArray. If a row of the query begins with a capital "T," that signifies that the day was a holiday with no stock trading and the loop skips that entry. To set the priceArray value for each row the code uses a combination of InStr and Len commands to parse out the "HIGH" price value on each row. To set

the timeArray values, the loop looks for rows that begin with a lowercase “a,” signifying the beginning of a new day of stock trading. It then resets a time counter back to 7:30AM, to be increased by one minute every time the code loops. The dayArray values are incremented by one each time the code comes to a row beginning with “a.” This is accomplished with the following code:

```
'sets price array, day array, and time array for each observation
a = -15
For x = 1 To UBound(priceArray)
 If Left(getDataSheet.Cells(9 + x, 1), 1) <> "T" Then
        priceArray(x) = Mid(getDataSheet.Cells(9 + x, 1).Text, _
        InStr(1, getDataSheet.Cells(9 + x, 1).Text, ",") + 1, _
        InStr(InStr(1, getDataSheet.Cells(9 + x, 1).Text, ",") + 1, _
        getDataSheet.Cells(9 + x, 1).Text, ",") - _
        InStr(1, getDataSheet.Cells(9 + x, 1).Text, ",") - 1)
   If Left(getDataSheet.Cells(9 + x, 1), 1) = "a" Then
    a = a + 1
    t = TimeValue("07:30:00")
   End If
  dayArray(x) = a
  timeArray(x) = t
  t = t + TimeValue("00:01:00")
 End If
Next
```

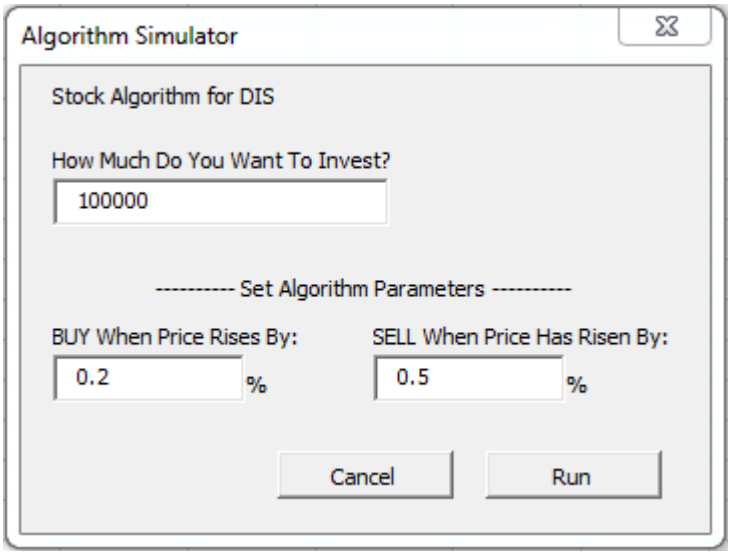
Next the sub calculates values for the changeArray based off the data in priceArray. Remember, changeArray is a *daily* percent change, so it resets every time that the dayArray changes. This is accomplished with the following code:

```
Dim b As Integer

'calculates values for changeArray
b = 1
For x = 2 To UBound(priceArray)
 If dayArray(x) <> dayArray(x - 1) Then
  b = x
  changeArray(x) = (priceArray(x) - priceArray(b)) / priceArray(b)
 Else:
  changeArray(x) = (priceArray(x) - priceArray(b)) / priceArray(b)
 End If
Next
```

Lastly, the sub clears out any existing data on the main worksheet and then pastes the new arrays:

```
'deletes any existing data and pastes new arrays
Range(Range("A5"), Range("A5").End(xlDown).End(xlToRight)).ClearContents

For x = 1 To UBound(priceArray)
 Sheet2.Cells(4 + x, 1).Value = dayArray(x)
 Sheet2.Cells(4 + x, 2).Value = timeArray(x)
 Sheet2.Cells(4 + x, 3).Value = priceArray(x)
 Sheet2.Cells(4 + x, 4).Value = changeArray(x)
Next

End Sub
```

User Form 1. This user form begins the algorithm simulation process. The user form appears as follows:



The user form contains three subroutines: (1) UserForm_Initialize, (2) CommandButton1_Click, and (3) CommandButton2_Click.

1. The UserForm_Initialize sub is activated via the "Algorithm Simulator" button on the custom ribbon. The sub first defines the caption for the user form. For example, if the user is simulating an algorithm for Disney stock, the caption on the user form will read "Stock Algorithm for DIS." The sub also loads the most recently used values (stored on a hidden sheet) as the default values into the three text boxes. This is accomplished with the following code:

```
Public Sub UserForm_Initialize()

 Label1.Caption = "Stock Algorithm for " & Sheet2.Range("B1").Value
 TextBox1.Value = Sheet1.Range("A2")
 TextBox2.Value = Sheet1.Range("B2")
 TextBox4.Value = Sheet1.Range("C2")

End Sub
```

2. The CommandButton1_Click sub corresponds to the "Run" button. This is where the algorithm calculations take place. This sub uses two of the same arrays that Module 1 uses (priceArray and changeArray) while defining two new arrays (principleArray and priceArrayNormal). They are created as follows:

```
Private Sub CommandButton1_Click()

Dim priceArray() As Double
Dim changeArray() As Single
Dim principleArray() As Double
Dim priceArrayNormal() As Double
Dim x As Integer

'redefines the arrays so that we can work with them inside the userform simulator
x = WorksheetFunction.CountA(Worksheets("getDataSheet").Range("A1:A10000")) - 9
 ReDim priceArray(x)
 ReDim changeArray(x)
 ReDim principleArray(x)
 ReDim priceArrayNormal(x)
For x = 1 To UBound(priceArray)
 priceArray(x) = Sheet2.Cells(4 + x, 3).Value
 changeArray(x) = Sheet2.Cells(4 + x, 4).Value
Next
```

Next, the variables “originalPrinciple”, “buy”, and “sell” are defined via the values in the textboxes. Buy and sell parameters correspond to percentage changes in the daily stock trading. To enter 1%, for example, the user inputs a 1 into the text box. To account for this, the sub later divides buy and sell parameters by 100 to make them true percentages.

```
Dim originalPrinciple As Double
Dim buy As Double
Dim sell As Double

originalPrinciple = TextBox1.Value
buy = TextBox2.Value
sell = TextBox4.Value
```

Next, the algorithm runs through every row of data via a For-Next loop. As it goes through each row, it assigns principleArray a value. This corresponds to how much money the user has (their principle) at each *minute* of each trading day. In order for the algorithm to buy stock, the following conditions must be met:

- Current value of stock is above the “buy” parameter [changeArray(x) >= buy / 100]
- Not currently invested [i=0]
- Current value of stock is below the “sell” parameter [changeArray(x) < sell / 100]
- Stock price is currently trending upward [changeArray(x - 1) < buy / 100]

```
'runs the simulator with provided parameters
Dim shares As Double
Dim i As Integer

principleArray(1) = originalPrinciple
For x = 2 To UBound(priceArray)
 If changeArray(x) >= buy / 100 Then
         If i = 1 Then
              If changeArray(x) > sell / 100 Then
               '4 SELL
               principleArray(x) = shares * priceArray(x)
               i = 0
               Else:
               '3 HOLD
               principleArray(x) = shares * priceArray(x)
              End If
         Else:
              If changeArray(x) < sell / 100 Then
              '2 BUY IF...
                    If changeArray(x - 1) < buy / 100 Then
                        '2.1 BUY because stock uptrending
                        principleArray(x) = principleArray(x - 1)
                        shares = principleArray(x) / priceArray(x)
                        i = 1
                    Else:
                        '2.2 DO NOT BUY because stock downtrending
                        principleArray(x) = principleArray(x - 1)
                    End If
              Else:
              '5 DO NOT BUY
              principleArray(x) = principleArray(x - 1)
              End If
        End If
  Else:
        If i = 1 Then
        '6 SELL
        principleArray(x) = shares * priceArray(x)
        i = 0
        Else:
        '1 DO NOT BUY
        principleArray(x) = principleArray(x - 1)
        End If
 End If
Next
```

If all four of these conditions are not met, the algorithm will either sell stock, hold stock, or not buy stock, depending on which conditions remain satisfied.

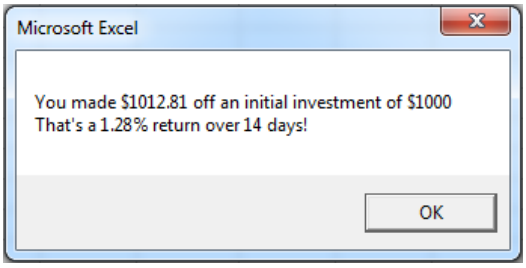
Next, the sub deletes any existing principle data off of the main worksheet and pastes values the newly defined principleArray:

```
'deletes any data and pastes principleArray

Range(Range("E5"), Range("E5").End(xlDown)).ClearContents

For x = 1 To UBound(priceArray)
 Sheet2.Cells(4 + x, 5).Value = principleArray(x)
Next
```

Next, the sub pastes the values from the textboxes onto a hidden sheet in order to use them as presets the next time the simulator is run:

```
'sets used values as presets for next time
Sheet1.Range("A2") = TextBox1.Value
Sheet1.Range("B2") = TextBox2.Value
Sheet1.Range("C2") = TextBox4.Value
```

Next, the sub hides the UserForm1 and presents the following message box:



Which is generated with the following code:

```
UserForm1.Hide

MsgBox "You made $" & Left(principleArray(UBound(principleArray)), 7) & _
" off an initial investment of $" & originalPrinciple & vbNewLine & "That's a " _
& Left((principleArray(UBound(principleArray)) - originalPrinciple) / originalPrinciple * 100, 4) _
& "% return over 14 days!"
```

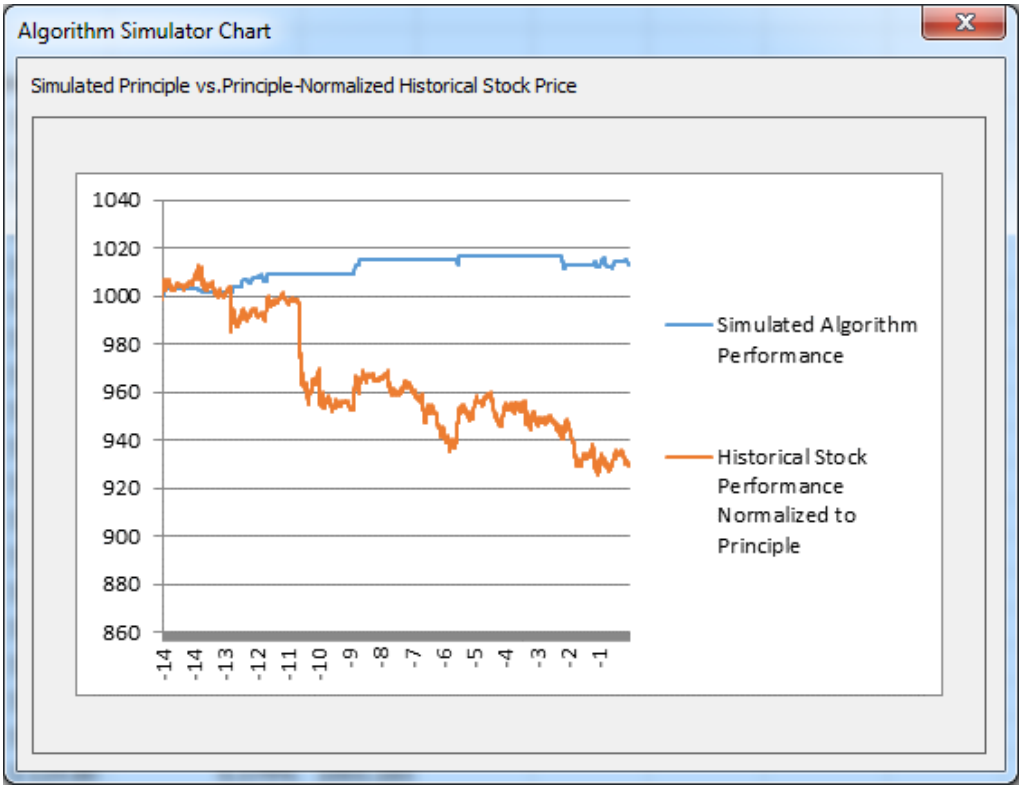
Lastly, before the UserForm1 closes and subsequently calls UserForm2, it generates values for an array called priceArrayNormal. This array simulates what the user's principle would have been if they had simply invested their principle on day 1 and held it over the 14 day period. It is created in order to benchmark the performance of the algorithm. It pastes the array values onto a hidden sheet, to be used by UserForm2, which will graph the algorithm's performance against priceArrayNormal. UserForm1 is then closed with the Unload Me command and calls UserForm2:

```
'deletes existing and sends data to presets page for historical data normalized to principle
Sheet1.Range(Sheet1.Range("A5"), Sheet1.Range("A5").End(xlDown)).ClearContents
Dim l As Double
 l = originalPrinciple / priceArray(1)
 For x = 1 To UBound(priceArray)
  priceArrayNormal(x) = priceArray(x) * l
  Sheet1.Cells(4 + x, 1).Value = priceArrayNormal(x)
 Next

Unload Me

UserForm2.Show

End Sub
```

3. The CommandButton2_Click sub corresponds to the cancel button and simply closes the user form:

```
Private Sub CommandButton2_Click()
Unload Me
End Sub
```

User Form 2. This user form provides a graph of the stock's performance vs. the algorithm simulator's performance. The user form appears as follows:



All of the code for UserForm2 is contained in one sub called UserForm_Initialize.

1. UserForm_Initialize first creates a line chart on the active sheet which is the main worksheet of the program (and only visible worksheet). Screen updating is off so the user never sees this happen. The sub then clears out the existing data series that the chart generated by default:

```
Private Sub UserForm_Initialize()

Dim myChart As Chart

Application.ScreenUpdating = False

'creates the line chart
Set myChart = ActiveSheet.Shapes.AddChart(xlLine).Chart

'clears default chart series
Dim n As Long
    With myChart
        For n = .SeriesCollection.Count To 1 Step -1
            .SeriesCollection(n).Delete
        Next n
    End With
```

Next, the sub add a two data series to the chart. One for the simulated algorithm performance and one for the historical stock performance:

```
'adds algorithm simulation series
myChart.SeriesCollection.NewSeries
myChart.SeriesCollection(1).Name = "Simulated Algorithm Performance"
myChart.SeriesCollection(1).Values = Range(Range("E5"), Range("E5").End(xlDown))
myChart.SeriesCollection(1).XValues = Range(Range("A5"), Range("A5").End(xlDown))

'adds benchmark historical stock price series
myChart.SeriesCollection.NewSeries
myChart.SeriesCollection(2).Name = "Historical Stock Performance Normalized to Principle"
myChart.SeriesCollection(2).Values = Sheet1.Range(Sheet1.Range("A5"), Sheet1.Range("A5").End(xlDown))
```

Lastly, the sub exports the chart as an image, then loads the image onto the user form:

```
'exports the chart as an image, then loads the image onto the user form
Dim imageName As String
imageName = Application.DefaultFilePath & Application.PathSeparator & "TempChart.gif"

myChart.Export Filename:=imageName
ActiveSheet.ChartObjects(1).Delete
Application.ScreenUpdating = True
UserForm2.Image1.Picture = LoadPicture(imageName)

End Sub
```

“Data” Worksheet.

After refreshing the stock data and simulating the algorithm, the user is left on the “Data” worksheet with all of the data needed for further analysis:

	A	B	C	D	E
1	Ticker:	DIS			
2					
3					
4	**Relative Trading Days**	**Time**	**Price**	**Percent Change**	**Principle**
5	-14	7:30:00 AM	$ 119.11	0.000%	1000
6	-14	7:31:00 AM	$ 119.27	0.134%	1000
7	-14	7:32:00 AM	$ 119.18	0.060%	1000
8	-14	7:33:00 AM	$ 118.98	-0.106%	1000
9	-14	7:34:00 AM	$ 119.03	-0.067%	1000
10	-14	7:35:00 AM	$ 119.18	0.062%	1000
11	-14	7:36:00 AM	$ 119.30	0.160%	1000
12	-14	7:37:00 AM	$ 119.34	0.193%	1000
13	-14	7:38:00 AM	$ 119.37	0.218%	1000
14	-14	7:39:00 AM	$ 119.46	0.293%	1000.75
15	-14	7:40:00 AM	$ 119.43	0.269%	1000.503
16	-14	7:41:00 AM	$ 119.41	0.252%	1000.335
17	-14	7:42:00 AM	$ 119.40	0.243%	1000.251
18	-14	7:43:00 AM	$ 119.40	0.247%	1000.287
19	-14	7:44:00 AM	$ 119.58	0.395%	1001.759
20	-14	7:45:00 AM	$ 119.55	0.366%	1001.476
21	-14	7:46:00 AM	$ 119.66	0.462%	1002.429
22	-14	7:47:00 AM	$ 119.75	0.537%	1003.183

Learning and Conceptual Difficulties Encountered

Substantial learning was required for me to make this program work the way it does.

- I learned that it is often easiest to flow chart your code on paper before attempting to write it. This was especially the case while I was writing my algorithm. I wasted a lot of time when I tried to dive right in and code it with little to no thought as to how it was going to work. I eventually drew out a large flow chart on paper that made my code much simpler to write.
- I learned that there are substantial resources available to VBA coders on the internet. I had to refer to the internet several times to accomplish my goals. For example, I found a blog post that explained how to put charts on user forms, which helped me quite a bit with UserForm2.
- I learned to be methodical and take my time when writing code. It is easier to write code correctly the first time than to write it incorrectly and waste 15 minutes trying to figure out your mistake.
- I learned that the user experience matters 100% of the time. By this I mean that a large portion of my time was spent finding ways to fix small bugs that only slightly affected the user experience. At first I tried to rationalize not fixing them saying things like, “this will only happen 1 out of 100 times the code runs.” In the end, as a user myself, I found these small bugs just as frustrating as the big bugs and so I fixed them.

I was able to include everything in my project that I wanted to.

Assistance

I did not receive substantial assistance on this project, although I consulted with Dr. Allen on a couple occasions and I consulted the internet and/or class textbook regularly.