Using VBA to Model the Forecast of a Bond's Cash Flows

Jacob Kunzler

Brigham Young University

Dr. Gove Allen

Information Systems 520

April 16, 2014

Executive Summary

Business description. Beneficial Financial Group is a life insurance company headquartered in Salt Lake City that is currently in "run-off"—the company no longer issues insurance policies, so it is gradually shrinking rather than growing. As such, Beneficial is currently focusing its efforts on servicing the policies it has already issued, which mainly entails investment and actuarial management. One of the reasons that Beneficial has gone into run-off is that mortgage-backed securities (MBS) had made up such a large proportion of its investment portfolio at the time the 2007-2008 financial crisis hit. After the almost-overnight drop in value of the MBS market in 2007, companies like Beneficial struggled to survive if they had been improperly assessing the risk of their MBS portfolio (as nearly the entire market had been). Since the crisis, Beneficial has been working to improve its procedures as much as possible to understand the true risk inherent in the mortgage-backed assets it holds.

Overview of problem and VBA solution. One of the procedures Beneficial is currently undertaking to assess the risk of its MBS portfolio is to dissect its mortgage-backed bonds into the expected cash flows of the underlying loans that back the security. This is done in an effort to better understand the individual loans that account for the profitability of the bond, something that had been largely neglected prior to the crisis. The process of dissecting these bonds, however, can be somewhat tedious. Each bond has loan data contained in a workbook unique to the bond. Beneficial keeps another workbook which contains a model for forecasting the cash flows of the loans underlying the bond. The firm keeps a third "master" workbook which opens the bond workbook, extracts the loan data, opens the model workbook, and exports the loan data into the model to forecast the cash flows.

In order to relinquish the need to keep track of so many workbooks, analysts at Beneficial are interested in simplifying the process by maintaining one workbook for each bond/model combination. They have sent me the workbook of a manufactured housing bond they hare holding, along with the model workbook and the master workbook. The master workbook contains a macro that had been previously written by analysts to calculate the principal and interest (P&I) from an individual bond's loan data, but no code had been written to convert the P&I into a forecast of cash flows classified by loan type (which they currently use the model workbook to create). I have written code which calls the macro that calculates the P&I, creates a cash flow forecast identical to the one created by the model workbook, and then pastes it into new sheets in the bond workbook. The code I've written eliminates the need for the model and master workbooks altogether. Furthermore, I've created a tab in the ribbon with buttons to simplify the analysis process, along with an add-in that Beneficial can use to access the model from any workstation.

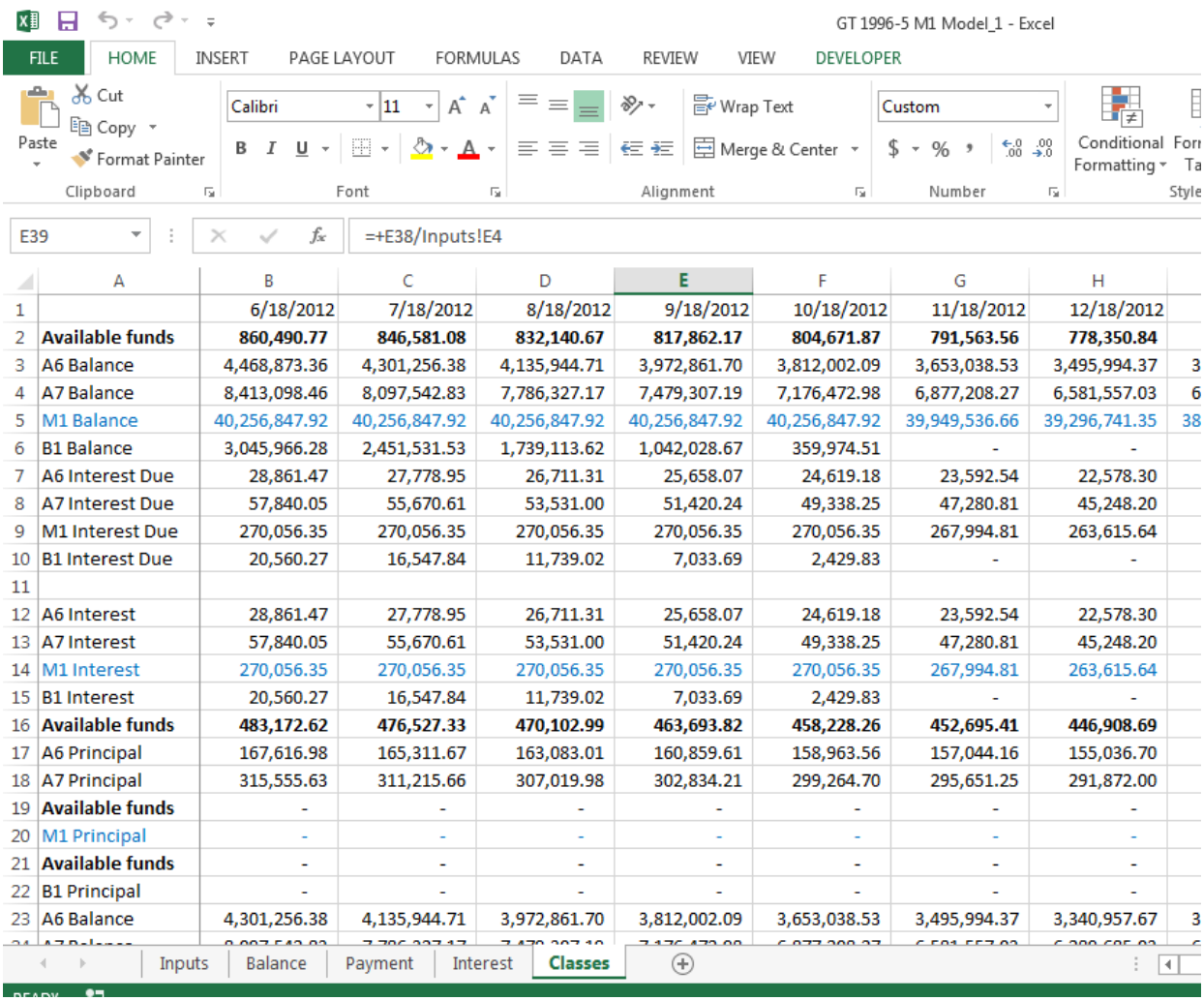
Implementation Documentation

The code I've written is divided into three parts: (1) calling the previously-written macro that extracts the loan data from the bond spreadsheet and calculates the P&I, (2) running the model which forecasts the loan cash flows by classification, and (3) exporting the data into new sheets in the bond's workbook. These parts are carried out by three distinct sub-procedures (*Sub CalcPAndI*, *Sub RunModel*, and *Sub PasteIntoWorkbook*) which are called by a fourth "master" sub-procedure (*Sub Master*). I wrote a fifth sub to delete the newly created sheets (*Sub DeleteSheets*), should the user decide to reset the workbook to its original state.

(1) *Sub CalcPAndI ()* was written by Beneficial and was already contained in the master spreadsheet that was sent to me. Besides cleaning it up a bit and labeling the code, I didn't take much part in creating this portion of the process. I copied the code into the bond's workbook, and went through line by line to ensure that I understood the process and the variables. This code would play a critical role in the code I would then write in the subsequent sub-procedures. *Sub CalcPAndI* uses the values in the "Data" and "Inputs" sheets of the bond workbook to calculate the principal and interest forecasts that will be dissected by class in *Sub RunModel.*

(2) Creating *Sub RunModel* was a much more involved process, and required understanding the model workbook and all of its functions, and then replicating them in VBA. Figure 1 shows the model I would need to recreate.

Figure 1 – Original model spreadsheet (not in VBA)



	A	B	C	D	E	F	G	H
1		6/18/2012	7/18/2012	8/18/2012	9/18/2012	10/18/2012	11/18/2012	12/18/2012
2	**Available funds**	**860,490.77**	**846,581.08**	**832,140.67**	**817,862.17**	**804,671.87**	**791,563.56**	**778,350.84**
3	A6 Balance	4,468,873.36	4,301,256.38	4,135,944.71	3,972,861.70	3,812,002.09	3,653,038.53	3,495,994.37
4	A7 Balance	8,413,098.46	8,097,542.83	7,786,327.17	7,479,307.19	7,176,472.98	6,877,208.27	6,581,557.03
5	M1 Balance	40,256,847.92	40,256,847.92	40,256,847.92	40,256,847.92	40,256,847.92	39,949,536.66	39,296,741.35
6	B1 Balance	3,045,966.28	2,451,531.53	1,739,113.62	1,042,028.67	359,974.51	-	-
7	A6 Interest Due	28,861.47	27,778.95	26,711.31	25,658.07	24,619.18	23,592.54	22,578.30
8	A7 Interest Due	57,840.05	55,670.61	53,531.00	51,420.24	49,338.25	47,280.81	45,248.20
9	M1 Interest Due	270,056.35	270,056.35	270,056.35	270,056.35	270,056.35	267,994.81	263,615.64
10	B1 Interest Due	20,560.27	16,547.84	11,739.02	7,033.69	2,429.83	-	-
11								
12	A6 Interest	28,861.47	27,778.95	26,711.31	25,658.07	24,619.18	23,592.54	22,578.30
13	A7 Interest	57,840.05	55,670.61	53,531.00	51,420.24	49,338.25	47,280.81	45,248.20
14	M1 Interest	270,056.35	270,056.35	270,056.35	270,056.35	270,056.35	267,994.81	263,615.64
15	B1 Interest	20,560.27	16,547.84	11,739.02	7,033.69	2,429.83	-	-
16	**Available funds**	**483,172.62**	**476,527.33**	**470,102.99**	**463,693.82**	**458,228.26**	**452,695.41**	**446,908.69**
17	A6 Principal	167,616.98	165,311.67	163,083.01	160,859.61	158,963.56	157,044.16	155,036.70
18	A7 Principal	315,555.63	311,215.66	307,019.98	302,834.21	299,264.70	295,651.25	291,872.00
19	**Available funds**	-	-	-	-	-	-	-
20	M1 Principal	-	-	-	-	-	-	-
21	**Available funds**	-	-	-	-	-	-	-
22	B1 Principal	-	-	-	-	-	-	-
23	A6 Balance	4,301,256.38	4,135,944.71	3,972,861.70	3,812,002.09	3,653,038.53	3,495,994.37	3,340,957.67

In order to carry out the process, I created nine array variables to work with, eight of which were two-dimensional. Because most of the variables in the model are divided into four

classifications (e.g., A6 Interest, A7 Interest, M1 Interest, B1 Interest), one of the dimensions in these variables was assigned to this classification (e.g., classInt(1,x) refers to A6 Interest, classInt(2,x) refers to A7 Interest, etc.). The second dimension had a value between 1 and 360, in accordance with the arrays created in *Sub CalcPAndI* that are used in this sub.

Sub RunModel is essentially one massive For loop that goes row-by-row through each column and calculates values for each of the variables based on the values of previously calculated variables. Some of the calculations can become quite complex, and involve a series of additional nested For loops, If statements, and nested If statements. An understanding of the logic inherent in the model spreadsheet was critical to creating the logic that would obtain the exact same values for the arrays in the VBA. Figure 2 provides a sample of some of the calculations required to run the model.

Figure 2 – Sample of code from Sub RunModel

```
'get date
Dates(x) = DateAdd("m", 1, Dates(x - 1))

'calc available funds
AvailFunds(1, x) = (1 - Del) * (PrincipalAgg(x) + InterestAgg(x))

'calc first balance and interest due for each class
For y = 1 To 2
    If x = 1 Then
        classBalance1(y, x) = Sheets("Inputs").Range("ClassCurrAmt").Offset(y, 0).Value
        Else:
        classBalance1(y, x) = classBalance2(y, x - 1)
    End If
    classIntDue(y, x) = Sheets("Inputs").Range("ClassCpn").Offset(y, 0).Value / 12 * classBalance1(y, x)
Next y
For y = 3 To 4
    If x = 1 Then
        classBalance1(y, x) = Sheets("Inputs").Range("ClassCurrAmt").Offset(y, 0).Value
        Else:
        classBalance1(y, x) = classBalance2(y, x - 1) - classLoss(y, x - 1)
    End If
    classIntDue(y, x) = Sheets("Inputs").Range("ClassCpn").Offset(y, 0).Value / 12 * classBalance1(y, x)
Next y
```

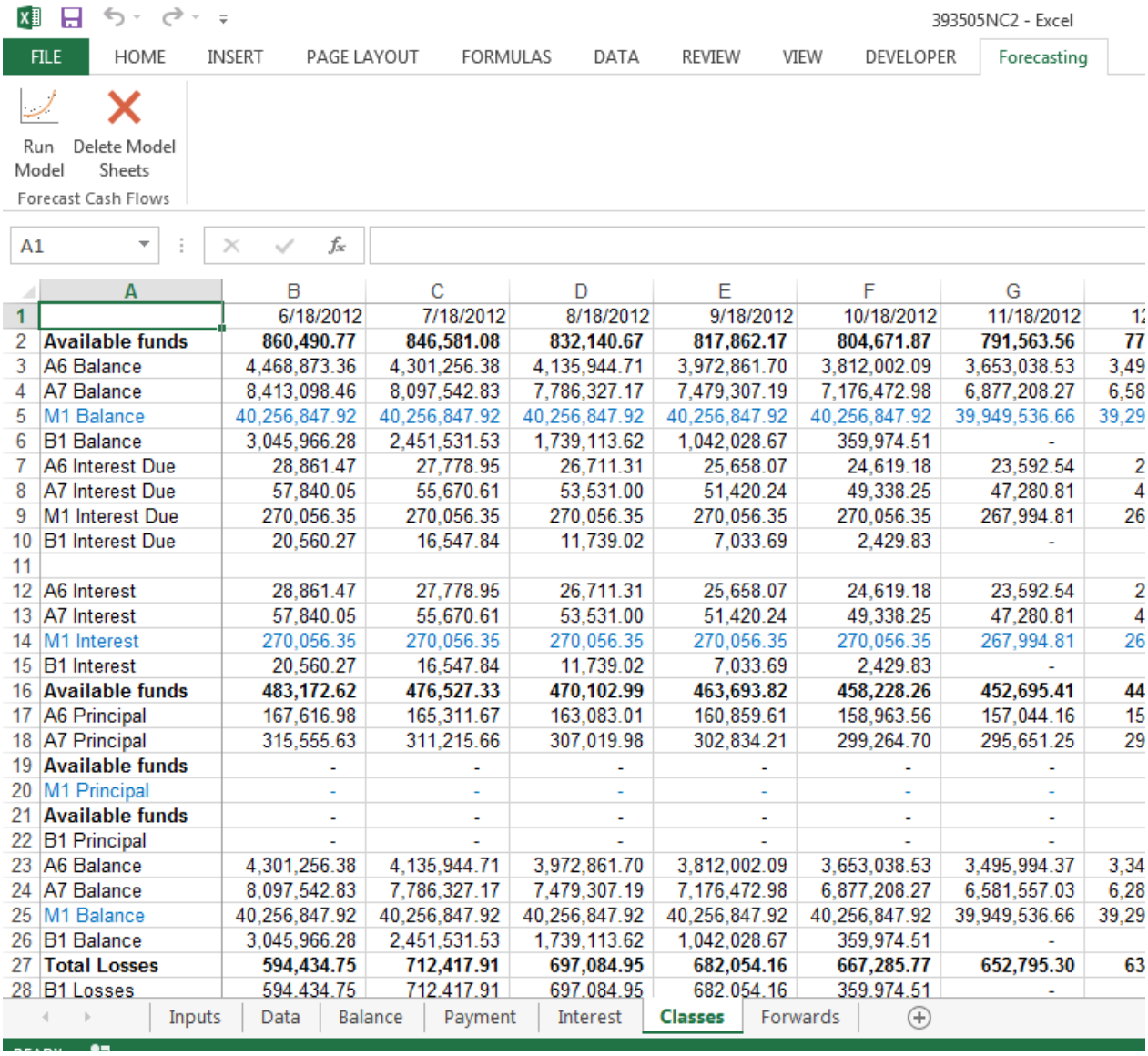
(3) *Sub PasteIntoWorkbook* is divided into three steps: (1) delete any sheets used to forecast the model's cash flows (if the sheets already exist), (2) create new sheets to display the model's forecast, (3) paste the values calculated in *Sub RunModel*, and (4) format the sheets to look like the original model. The formatting included freezing the panes, auto-fitting the columns, coloring some of the "M1" forecasts in blue, changing the number format, and bolding the "Available Funds" and all "Losses" rows.

(4) *Sub Master* runs the three aforementioned subs in order. I created a "Forecasting" tab in the ribbon to display a button entitled "Run Model" in the "Forecast Cash Flows" group, which executes the *Sub Master* macro.

(5) *Sub DeleteSheets* deletes the worksheets created by the *PasteIntoWorkbook* sub, in order to return the workbook to its original state. A button for this sub is also located in the "Forecast Cash Flows" group of the newly created "Forecasting" tab in the ribbon.

Figure 3 shows the new "Forecasting" tab of the ribbon, along the projected cash flows, divided by classification, that are created by running the system of macros.

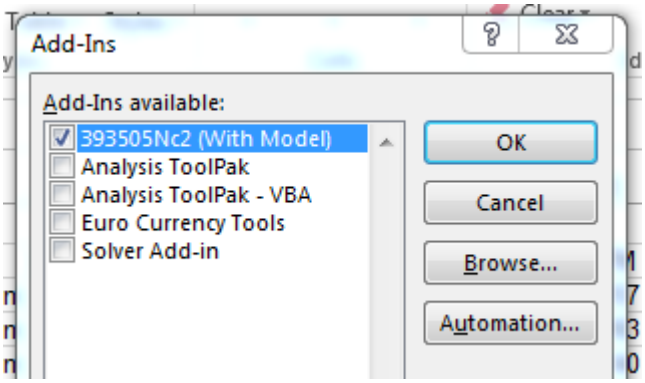
Figure 3 – "Forecasting" tab of the ribbon and model outputs (from VBA)



	A	B	C	D	E	F	G	
1		6/18/2012	7/18/2012	8/18/2012	9/18/2012	10/18/2012	11/18/2012	1
2	**Available funds**	**860,490.77**	**846,581.08**	**832,140.67**	**817,862.17**	**804,671.87**	**791,563.56**	**77**
3	A6 Balance	4,468,873.36	4,301,256.38	4,135,944.71	3,972,861.70	3,812,002.09	3,653,038.53	3,49
4	A7 Balance	8,413,098.46	8,097,542.83	7,786,327.17	7,479,307.19	7,176,472.98	6,877,208.27	6,58
5	M1 Balance	40,256,847.92	40,256,847.92	40,256,847.92	40,256,847.92	40,256,847.92	39,949,536.66	39,29
6	B1 Balance	3,045,966.28	2,451,531.53	1,739,113.62	1,042,028.67	359,974.51	-	
7	A6 Interest Due	28,861.47	27,778.95	26,711.31	25,658.07	24,619.18	23,592.54	2
8	A7 Interest Due	57,840.05	55,670.61	53,531.00	51,420.24	49,338.25	47,280.81	4
9	M1 Interest Due	270,056.35	270,056.35	270,056.35	270,056.35	270,056.35	267,994.81	26
10	B1 Interest Due	20,560.27	16,547.84	11,739.02	7,033.69	2,429.83	-	
11								
12	A6 Interest	28,861.47	27,778.95	26,711.31	25,658.07	24,619.18	23,592.54	2
13	A7 Interest	57,840.05	55,670.61	53,531.00	51,420.24	49,338.25	47,280.81	4
14	M1 Interest	270,056.35	270,056.35	270,056.35	270,056.35	270,056.35	267,994.81	26
15	B1 Interest	20,560.27	16,547.84	11,739.02	7,033.69	2,429.83	-	
16	**Available funds**	**483,172.62**	**476,527.33**	**470,102.99**	**463,693.82**	**458,228.26**	**452,695.41**	**44**
17	A6 Principal	167,616.98	165,311.67	163,083.01	160,859.61	158,963.56	157,044.16	15
18	A7 Principal	315,555.63	311,215.66	307,019.98	302,834.21	299,264.70	295,651.25	29
19	**Available funds**	-	-	-	-	-	-	
20	M1 Principal	-	-	-	-	-	-	
21	**Available funds**	-	-	-	-	-	-	
22	B1 Principal	-	-	-	-	-	-	
23	A6 Balance	4,301,256.38	4,135,944.71	3,972,861.70	3,812,002.09	3,653,038.53	3,495,994.37	3,34
24	A7 Balance	8,097,542.83	7,786,327.17	7,479,307.19	7,176,472.98	6,877,208.27	6,581,557.03	6,28
25	M1 Balance	40,256,847.92	40,256,847.92	40,256,847.92	40,256,847.92	40,256,847.92	39,949,536.66	39,29
26	B1 Balance	3,045,966.28	2,451,531.53	1,739,113.62	1,042,028.67	359,974.51	-	
27	**Total Losses**	**594,434.75**	**712,417.91**	**697,084.95**	**682,054.16**	**667,285.77**	**652,795.30**	**63**
28	B1 Losses	594,434.75	712,417.91	697,084.95	682,054.16	359,974.51	-	

Model Add-In. To simplify the process of collaboration among the analysts at Beneficial, I created an add-in which can be installed at the analysts' work stations. The add-in will create the "Forecasting" ribbon tab in Excel, regardless of whether the workbook I've written the code into is accessible by them. Now, rather than model new bonds into individual worksheets, bonds can be written into code and put into this ribbon as a new button, and whichever bond workbook is opened, a corresponding button can be displayed to model the bond's forecasted cash flows.

Figure 4 – Add-in for the bond's forecasting model



Learning and Difficulties Encountered

I originally thought that this project would be rather straightforward, with a clear path in sight. As I got involved in the coding process, however, I quickly realized that a series of ad-hoc puzzles would dominate the course of the project. Dealing with these unforeseen problems led to substantial learning and growth. Some of the most significant insights I gained and problems I overcame are outlined below.

- *Public vs. private variables.* Prior to the project, I had not worked much with public variables, mostly because I had never needed to. The nature of this project required me to use variables in more than one sub-procedure, and they needed to retain their values from one sub to the next. Moreover, there were other variables with values that I did *not* want to retain, and I had to ensure that they were kept private from other subs. I learned to utilize *Option Explicit* so as to avoid making mistakes in the large numbers of variables I would need to work with in the forecasting process.

- *Model organization.* One of the biggest problems I encountered was the difficulty of replicating a disorganized spreadsheet model in VBA. Things I had originally thought would fit neatly into loops often did not; much of the time, one classification required information that others did not, and sometimes the first dimension of one array would be increasing as the first dimension of another array would be decreasing. Figure 5 shows an example of a workaround I had to create for one such problem. In order to simultaneously step forwards in one array as I stepped backwards through a For loop for another array, I had to create a logical solution of adding -1 times the step variable to a positive number.

Figure 5 – Solution to a tricky problem with increasing and decreasing dimensions of an array

```
For y = 3 To 1 Step -1
    If remainLoss(-y + 4, x) > 0 Then
        classLoss(y, x) = Min(remainLoss(-y + 4, x), classBalance2(y, x))
        Else:
            classLoss(y, x) = 0
    End If
    If Not y = 1 Then remainLoss(-y + 5, x) = remainLoss(-y + 4, x) - classLoss(y, x)
Next y
```

- *Troubleshooting/debugging.* Possibly the most time-consuming and mentally aggravating of the difficulties encountered in this project was simply getting the macro to produce the right numbers. The tiniest logical error in one step of the sub would result in incorrect calculations for the rest of the 12,000 values following it. After all the problems with the logic had finally been weeded out, and the output from the code at last matched up with the numbers in the model workbook, it felt like an absolute miracle.

- *Working with arrays.* I felt that this project forced me to develop a fluency in dealing with array variables, and two-dimensional arrays in particular. I learned to picture invisible spreadsheets in my head, with rows and columns pertaining to the dimensions of the array. The combination of overarching and nested For loops helped me to envision that one of the dimensions was stepping as the nested loops were stepping, and the other dimension stepped as the overarching loop stepped.

- *Formatting.* This project also helped me to learn the code for changing font color, auto-fitting the columns, setting a freeze pane, formatting numbers, and manipulating other factors that I hadn't needed to alter previously. I learned which methods corresponded with which objects (e.g., the worksheet will not execute the .AutoFit method—the "Columns" property of a range must be referenced).

Assistance Received

The code written in the *CalcPAndI* sub-procedure—which extracts the loan data from the bond workbook and uses it to calculate the loans' P&I—was entirely provided by Beneficial. In addition, the spreadsheet model used to forecast the cash flows was created by Beneficial analysts from the bond's prospectus. Creating such a model from a bond's prospectus is beyond the scope of what I am currently capable. I used Beneficial's model spreadsheet to understand the logic I would need in order to forecast the bond's cash flows in my code. All of the VBA code beyond *Sub CalcPAndI* is of my own creation.