BOND_PORT_ANALYSIS_FUNC TECHNICAL REPORT

Rafael Nicolas Fermin Cota

Abstract

This Technical Report provides a full line by line explanation of the BOND_PORT_ANALYSIS_FUNC function.

Introduction to the Function

About this Report

This report provides a line by line explanation of the BOND_PORT_ANALYSIS_FUNC function algorithm. It is meant to give the reader a detailed explanation of how the function operates and produces its outputs.

About The Function

The BOND_PORT_ANALYSIS_FUNC function possesses an extremely comprehensive and dynamic algorithm to assist its user in bond portfolio management. The function produces a wide selection of outputs and provides information on both the individual bonds in the portfolio, and the overall portfolio. The function allows the user to perform sensitivity analysis on the portfolio by adding and taking away bonds from the portfolio to analyze the overall effect.

The function is dynamic and allows the user the option to input more variables above what is required to further enhance the analysis generated with just the required inputs. The function will also automatically adjust its output matrix to any changes made, whether it be by adding more bonds to the portfolio or adding an optional input.

Setting-Up the Function

Function Variables

Input Variable

Variable Name	Required/Optional	Type	Purpose
VALOR_RNG	Required	Variant	Input Variable
MOODY_RATING_RNG	Required	Variant	Input Variable
SP_RATING_RNG	Required	Variant	Input Variable
INSURANCE_NAME_RNG	Required	Variant	Input Variable
CUSIP_RNG	Required	Variant	Input Variable
ISSUE_NAME_RNG	Required	Variant	Input Variable
BOND_TYPE_RNG	Required	Variant	Input Variable
COUPON_RNG	Required	Variant	Input Variable
FREQUENCY_RNG	Required	Variant	Input Variable
MATURITY_RNG	Required	Variant	Input Variable
CALL_PRICE_RNG	Required	Variant	Input Variable
CALL_DATE_RNG	Required	Variant	Input Variable
PURCHASE_DATE_RNG	Required	Variant	Input Variable
PURCHASE_PRICE_RNG	Required	Variant	Input Variable
PURCHASE_QUANTITY_RNG	Required	Variant	Input Variable
PURCHASE_FEES_RNG	Required	Variant	Input Variable
CURRENT_PRICE_RNG	Required	Variant	Input Variable
MARKET_YIELDS_RNG	Optional	Variant	Input Variable
CREDIT_AGENCIES_RNG	Optional	Variant	Input Variable
PORT_BOND_TYPES_RNG	Optional	Variant	Input Variable
PORT_MATURITIES_RNG	Optional	Variant	Input Variable
PORT_CREDIT_QUALITIES_RNG	Optional	Variant	Input Variable

DELTA_VAL	Optional	Double	100 * Basis Points
REBALANCING_SLACK	Optional	Double	Used in Bond Maturity, Bond Type and Credit Quality. This value creates some "slack" for the rebalancing algorithm (between desired percent and actual percent) to reduce clutter in cases where the difference between the target percent allocation and the current percent allocation is small.
SETTLEMENT	Optional	Date	Date to base bond price/yield/ maturity calculations.
GUESS_YIELD	Optional	Variant	Used to solve for YTM.
COUNT_BASIS	Optional	Variant	Used in bond calculation functions in the algorithm.
OUTPUT	Optional	Integer	Defines what output the user wants.

Code Variables

Variable Name	Type	Purpose
h()	Long	Assist in making output dynamic
i	Long	Counter Variable
j	Long	Counter Variable
k	Long	Counter Variable
l	Long	Counter Variable
m	Long	Counter Variable
n	Long	Counter Variable
hh	Long	Counter Variable
ii	Long	Counter Variable
jj	Long	Counter Variable
kk	Long	Counter Variable
ll	Long	Counter Variable
mm	Long	Counter Variable
nn	Long	Counter Variable
NROWS	Long	Holds number of rows for TEMP0_MATRIX
NCOLUMNS	Long	Holds number of columns for TEMP0_MATRIX
Y_VAL	Double	Variable used for Yield and Price Calculations
P_VAL	Double	Variable used for Yield and Price Calculations
S_VAL	Date	Variable used for Yield and Price Calculations
M_VAL	Date	Variable used for Yield and Price Calculations
C_VAL	Double	Variable used for Yield and Price Calculations
F_VAL	Integer	Variable used for Yield and Price Calculations
R_VAL	Double	Variable used for Yield and Price Calculations
B_VAL	Integer	Variable used for Yield and Price Calculations
G_VAL	Double	Variable used for Yield and Price Calculations

NA_STR	String	Reports value “N/A”
NR_INT	Integer	Holds the value for a bond that is not rated
NR_STR	String	Holds Value “NR”
FIND_STR	String	Returns a value that is being searched
LOOK_STR	String	Holds a value that is used to find another value
MULT1_VAL	Constant As Double	Value is 10 to convert 100 prices into $1000 multiple
MULT2_VAL	Constant As Double	Value is 100 to convert coupon rate to percent
MULT3_VAL	Constant As Double	Value is 1000 to adjust quantity to $1000 multiple
HEADINGS0_STR	String	Used in Header Line
HEADINGS1_STR	String	Used in Header Line
HEADINGS_ARR()	String	Used in Header Line
TEMP0_VAL	Variant	Holds temporary values
TEMP1_VAL	Variant	Holds temporary values
TEMP2_VAL	Variant	Holds temporary values
DATE1_VAL	Date	Holds date values
DATE2_VAL	Date	Holds date values
TEMP0_MATRIX	Variant	Stores output for Case 0
TEMP1_MATRIX	Variant	Stores output for Case 1
TEMP2_MATRIX	Variant	Stores output for Case 2
TEMP3_MATRIX	Variant	Stores output for Case 3
TEMP4_MATRIX	Variant	Stores output for Case 4
PORT_BOND_TYPES_FLAG	Boolean	Determines if optional value for Bond Types was inputted
PORT_BOND_TYPES_MATRIX	Variant	Store information from Bond Type Input
PORT_MATURITIES_FLAG	Boolean	Determines if optional value for Maturities was inputted
PORT_BOND_MATURITIES_MATRIX	Variant	Store information from Maturities Input
PORT_CREDIT_QUALITIES_FLAG	Boolean	Determines if optional value for Credit Quality was inputted
PORT_CREDIT_QUALITIES_MATRIX	Variant	Stores information from Credit Quality Input
CREDIT_AGENCIES_FLAG	Boolean	Determines if optional value for Credit Agencies was inputted
CREDIT_AGENCIES_MATRIX	Variant	Stores information from Credit Agencies Input
MARKET_YIELDS_FLAG	Boolean	Determines if optional value for Market Yields was inputted
MARKET_YIELDS_MATRIX	Variant	Stores information from Market Yields Input
VALOR_VECTOR	Variant	Holds information for its respective input Range
MOODY_RATING_VECTOR	Variant	Holds information for its respective input Range
SP_RATING_VECTOR	Variant	Holds information for its respective input Range
INSURANCE_NAME_VECTOR	Variant	Holds information for its respective input Range
CUSIP_VECTOR	Variant	Holds information for its respective input Range
ISSUE_NAME_VECTOR	Variant	Holds information for its respective input Range
BOND_TYPE_VECTOR	Variant	Holds information for its respective input Range

COUPON_VECTOR	Variant	Holds information for its respective input Range
FREQUENCY_VECTOR	Variant	Holds information for its respective input Range
MATURITY_VECTOR	Variant	Holds information for its respective input Range
CALL_PRICE_VECTOR	Variant	Holds information for its respective input Range
CALL_DATE_VECTOR	Variant	Holds information for its respective input Range
PURCHASE_DATE_VECTOR	Variant	Holds information for its respective input Range
PURCHASE_PRICE_VECTOR	Variant	Holds information for its respective input Range
PURCHASE_QUANTITY_VECTOR	Variant	Holds information for its respective input Range
PURCHASE_FEES_VECTOR	Variant	Holds information for its respective input Range
CURRENT_PRICE_VECTOR	Variant	Holds information for its respective input Range
COUNT_BASIS_VECTOR	Variant	Holds information for its respective input Range
GUESS_YIELD_VECTOR	Variant	Holds information for its respective input Range
tolerance	Double	Used in optimization for calculating yields
epsilon	Double	Used to help calculate the price of a bond

Error Handling

If any error is triggered during the course of the function the code is sent to line ERROR_LABEL and the function outputs the error number.

Constant Variables

Some variables used in the function are pre-defined before the function begins. These include:

NA_STR is set equal to "N/A"
NR_STR is set equal to "NR"
tolerance is set equal to 10 ^ -15
epsilon is set equal to 0.0000001

For the SETTLEMENT variable, if it is set to zero then the date is set to NOW() or todays date. Otherwise it is set to whatever date is specified.

Passing the Inputs

Required Inputs

The code begins by passing all of the function's required inputs into their respective matrix.

The code for passing these required inputs is the same for all variables. Once the input is passed into its respective matrix the code checks the number of rows in that matrix. If there is only one row then the matrix is transposed to have its new number of rows equal to the original number of columns. The data held in the matrix would now be in one column with multiple rows as opposed to multiple columns and one row. If there is more than one row in the original input then nothing is changed.

During this process, the variable NROWS is set using the number of rows in ISSUE_NAME_RNG. This should intuitively make sense since the ISSUE_NAME_RNG holds the names of the bonds held in the portfolio. Therefore, the number of rows in the array will equal the number of securities in the portfolio.

Optional Inputs

Two optional variables are also defined in the process of passing the required inputs. These are the COUNT_BASIS_VECTOR and the GUESS_YIELD_VECTOR. While, these two inputs are optional for the user, they are still required for the function. This means that if they are not defined in the function inputs, then default values are needed.

Both of the optional inputs follow the same logic for determining if default values are needed. Three if-statements are used in the process.

The first if-statement checks if the input is an array. If it is then the input is set equal to the respective matrix.

The second if-statement, which is imbedded in the first, then performs the same condition check used for the required inputs. If the number of rows is equal to one then the matrix is transposed.

If in the first if-statement the input is not an array then the code jumps to the third if-statement to input default values. For the COUNT_BASIS variable if the input is blank and for the GUESS_YIELD variable if the input is blank or equal to zero, the code dimensions the respective matrix to equal NROWS and then uses a loop to input the default value into each of the rows.

The default values are 0 and 0.5 for the COUNT_BASIS and GUESS_YIELD_VECTOR respectively.

Dynamic Inputs

One of the unique aspects of the BOND_PORT_ANALYSIS_FUN function is its dynamic capabilities. The output of the function is dependent on three optional input variables: PORT_MATURITIES_RNG, PORT_BOND_TYPES_RNG and PORT_CREDIT_QUALITIES_RNG. The function then uses the index variable *h* to control the size of the output matrix.

This section of the code begins by re-dimensioning the index variable *h* to have 4 rows in base zero: h(0), h(1), h(2) and h(3). Next h(0) is set to equal 76. h(0) represents the minimum number of columns that the function will output regardless of the other optional input variables.

The function then checks to see if the optional inputs were inputted by the user. It does this by using the IsArray function. If the optional input was inputted into the function then it is passed to its respective matrix.

Each of the optional inputs also has a FLAG variable, which identifies whether or not data is present. If data is present then the FLAG is turned on, if not then the FLAG is turned off.

The first optional variable to be passed through these conditions is the MARKET_YIELDS_RNG. If data was inputted then the data is passed into the MARKET_YIELDS_MATRIX and the MARKET_YIELDS_FLAG is activated.

For the next three optional inputs: PORT_MATURITIES_RNG, PORT_BOND_TYPES_RNG and PORT_CREDIT_QUALITIES_RNG the same logic applies but with an added condition.

Since these three variables control the index variable *h* and thus the output , values need to be assigned to h(1), h(2) and h(3).

If data was inputted into the optional variable then the respective *h* will be set equal to the number of rows in the corresponding input. If no data was inputted then the respective *h* will be set to zero. The h values will then represent the added number of columns in each additional section needed as well as the original 76. The table below summarizes the logic.

Optional Variable	**h**	**h value if optional variable is present**	**h value if optional variable is not present**
PORT_MATURITIES_RNG	h(1)	Number of rows in PORT_CREDIT_QUALITIES_RNG	0
PORT_BOND_TYPES_RNG*	h(2)	Number of rows in PORT_BOND_TYPES_RNG	0
PORT_CREDIT_QUALITIES_RNG	h(3)	Number of rows in PORT_CREDIT_QUALITIES_RNG	0

* Specific to this case variable *j* is set equal to h(2)

Finally, the last optional variable checked in the function is the CREDIT_AGENCIES_RNG. If information was entered into the function for this variable then the CREDIT_AGENCIES_FLAG is turned on and the information is passed into the respective matrix. A loop is used to clean the information using the Trim function to remove spaces, and is then placed it back into the matrix in the same way it was before.

The function then checks to see if the last entry in the CREDIT_AGENCIES_MATRIX is numeric. If it is then the variable NR_INT is set equal to that number in the last row of CREDIT_AGENCIES_MATRIX. If it is not numeric or if no data was inputted for CREDIT_AGENCIES_RNG then the value of NR_INT is set to 8.

The NR_INT corresponds to the last Tier in the bond rating- Not Rated. The number that corresponds to this rating is 8. This is why the last row of CREDIT_AGENCIES_RNG is used to get the value for NR_INT (because it's the lowest rating) and why the default is 8.

Headings Line

Headings Line Part One

The headings line has two main purposes. As one might expect, it loads all of the required headings into the output matrix TEMP0_MATRIX and by doing this also determines the number of columns required in TEMP0_MATRIX.

This section begins by determining if the string HEADINGS0_STR is blank. If it is then the code continues. If it is not then the code jumps to line 1985, which is explained later.

Next NCOLUMNS is set equal to 19. This number represents the maximum number of sections the function can output assuming all the optional variables described above are inputted.

To fully understand how the headings work, it is first important to understand how the output is displayed. This function can generate a maximum of 19 sections. However, within each section there are multiple columns.

At this point the reader might question then why the variable NCOLUMNS is set equal to 19 if 19 represents the sections and not necessarily the number of columns.

The answer is that at this point the function does not know how many columns there will be as this is dependent on how many of the optional variables where inputted and also how many data points are in each optional variable.

To account for this the headings are first loaded in as a string of text into their respective section. For headings that are dependent on the optional variables, conditions are made so that if they are not present the column headings don't get loaded into that section.

Looking back at the code, after NCOLUMNS has been defined HEADINGS_ARR is re-dimensioned to have number of rows equal to NCOLUMNS. At this point each row in HEADINGS_ARR represents a section in the output.

Next a loop proceeds to insert blank spaces into each section and then the filling of each section begins.

Regardless of the optional variables discussed above, sections 1 through 5 and section 19 will always be present. The string of heading's names for each column in the section is loaded into their respective row in HEADINGS_ARR separated by a comma.

The remaining sections are all dependent on the optional variables discussed above. The logic for loading the headings for all the sections is the same.

For this part of the code the variable k is set equal to h(1) or h(2) or h(3) depending on the section being loaded into HEADINGS_ARR. Remember that these h values will equal the number of data points inputted for their respective optional variable.

A loop is then set up from 1 to k. If k = 0 then the loop will not proceed and the headings will be skipped. If k is not zero then the loop will continue.

For the sections that are dependent on the optional variable, the number of columns in each section is dependent on the number of data points passed through in the optional variable.

To create each heading the loop first identifies the value in the optional variable matrix that will be used in the column headings and assigns that value to the variable LOOK_STR. The loop then loops through the relevant sections and attaches the LOOK_STR variable into the column string. The next time through the loop, a new LOOK_STR is used and a new heading is created using that LOOK_STR. All of the headings in a given section are separated using a comma and stored in their respective row in HEADINGS_ARR.

For example:

If in the PORT_BOND_MATURITIES_MATRIX two values existed, 1 and 2, then in section six of the output- VALUE BY MATURITY BRACKET, two columns would be generated:

VALUE BY MATURITY BRACKET: 1 YEARS

VALUE BY MATURITY BRACKET: 2 YEARS

The code would read

… LOOK_STR = Format (PORT_BOND_MATURITIES_MATRIX(j, 2), "0.0")

HEADINGS_ARR(6) = HEADINGS_ARR(6) & "VI) VALUE BY MATURITY BRACKET: " & LOOK_STR & " Years" & "," …

Where LOOK_STR would equal the values 1 and 2; And HEADINGS_ARR(6) would house the two columns separated by a comma.

A summary of all the possible sections is given below:

Section	Name	Dependant on	Number of Columns
I	No Name	Not Dependant	24
II	No Name	Not Dependant	11
III	No Name	Not Dependant	12
IV	No Name	Not Dependant	12
V	No Name	Not Dependant	17
VI	VALUE BY MATURITY BRACKET:	PORT_MATURITIES_FLAG	h(1)
VII	VALUE BY BOND TYPE:	PORT_BOND_TYPES_FLAG	h(2)
VIII	VALUE BY CREDIT QUALITY:	PORT_CREDIT_QUALITIES_FLAG	h(3)
IX	CREDIT QUALITY BY BOND TYPE:	PORT_BOND_TYPES_FLAG	h(2)
X	CREDIT QUALITY BY MATURITY BRACKET:	PORT_MATURITIES_FLAG	h(1)
XI	MODIFIED DURATION BY BOND TYPE:	PORT_BOND_TYPES_FLAG	h(2)
XII	MODIFIED DURATION BY CREDIT QUALITY:	PORT_CREDIT_QUALITIES_FLAG	h(3)
XIII	PRE-TAX YIELD BY BOND TYPE:	PORT_BOND_TYPES_FLAG	h(2)
XIV	AFTER-TAX YIELD BY BOND TYPE:	PORT_BOND_TYPES_FLAG	h(2)
XV	PRE-TAX YIELD BY MATURITY BRACKET:	PORT_MATURITIES_FLAG	h(1)
XVI	AFTER-TAX YIELD BY MATURITY BRACKET:	PORT_MATURITIES_FLAG	h(1)
XVII	PRE-TAX YIELD BY CREDIT QUALITY:	PORT_CREDIT_QUALITIES_FLAG	h(3)
XVIII	AFTER-TAX YIELD BY CREDIT QUALITY:	PORT_CREDIT_QUALITIES_FLAG	h(3)

XIX	No Name	Not Dependant	12

Headings Line Part Two

Once all of the heading strings are loaded into their respective sections, the next part is to separate them into columns.

To start, HEADINGS0_STR is set equal to blank, and a loop begins to pass the contents of HEADINGS_ARR to HEADINGS0_STR to form one string of all the headings. Once this is complete HEADINGS_ARR is erased.

The code then moves to line 1985. The code that follows would be the code that would have proceeded if HEADINGS0_STR were not blank at the beginning of the function.

NCOLUMNS is then set equal to 0 and a loop begins to determine the actual number of columns that will appear in the output matrix.

It does this by counting the number of commas in the string HEADINGS0_STR. To do this a Do loop is set up which uses the InString function to originally start at position one (indicated by the i variable) and loop for the first comma in the string HEADINGS0_STR. InString returns the number of characters to the comma and is set equal to variable j. Every time the loop loops a tally is added to the NCOLUMNS variable. i is then set equal to j + 1 to start at the next position after the comma, and the loop begins again looking for the next comma in HEADINGS0_STR. This loop continues until no more commas can be found in the string. At this point j will equal 0 leading i to equal 1 and the loop ending.

The next line subtracts 1 from the NCOLUMS variable. This is because the loop above will always produce a value for NCOLUMS that has one column more than is needed. This is because in the loop above, even after no more commas are found, a tally is still added to NCOLUMNS before the loop exits.

TEMP0_MATRIX is then re-dimensioned to have 0 to NROWS + 1 number of rows and 1 to NCOLUMNS number of columns.

While the reasoning for the number of columns should be intuitive, the reader might wonder why the number of rows is set in base zero with a plus one. The reason why two more rows are added to NROWS is to account for an additional row for the headings, and an additional row for the Portfolio line, which is discussed in the next section of this report.

The final step in the process is to place all of the headings from HEADINGS0_STR into their appropriate column in the 0^{th} row of TEMP0_MATRIX.

This process is done using a loop similar to the Do loop used to determine the number of columns. This loop however loops from 1 to NCOLUMNS. It uses the same Instring function to perform the same operation of determining the number of characters from a starting position i to a comma. Once the number of characters is found it is stored in variable j. The mid function is then used to return the actual name of the heading being extracted. This is done by starting at the ith position and extracting j-i number of characters from HEADINGSO_STR. This process pulls each heading out of HEADING0_STR and places it in its appropriate column.

Once all the headings have been placed. A loop begins that places a blank space in all the remaining cells in the matrix TEMP0_MATRIX. The headings line is then complete.

Setting Up the Initial Portfolio Line

The last row of the function's output shows the user how the overall portfolio performed for most of the criteria in a respective column. This helps the user make decisions for the overall well being of the portfolio and also helps in sensitivity analysis.

This section begins by placing the name "Portfolio" in the last row and first column of the output matrix TEMP0_MATRIX.

Next zeros are placed into certain columns of the portfolio line. These columns include: column 6, 7, 30, 31, 36, 59 and 74.

hh is then set equal to h (0) or 76

ii is set equal to hh+1

jj = hh + h(1) + h(2) + h(3)

A loop is then used to place zeros into every column on the Portfolio line from ii or 77 to jj.

Next nn is set equal to hh + h(1) * 4 + h(2) * 5 + h(3) * 4 +1. The starting point of section 19 of the output. (By default h(0) has 5 sections)

The Portfolio line on column nn is then set equal to zero.

The Giant i Loop

Introduction to the Giant i Loop

This loop will loop from 1 to NROWS, or in other words for all of the data points in the inputs. NROWS also corresponds to the number of rows in the output.

Cleaning the Vector Data

Before the function proceeds, the following Vectors have their data cleaned using the Trim function, which removes any unnecessary spaces from the data. These vectors include:

- VALOR_VECTOR(i, 1)
- CURRENT_PRICE_VECTOR(i, 1)
- MATURITY_VECTOR(i, 1)
- PURCHASE_DATE_VECTOR(i, 1)
- PURCHASE_PRICE_VECTOR(i, 1)
- PURCHASE_QUANTITY_VECTOR(i, 1)
- PURCHASE_FEES_VECTOR(i, 1)
- COUPON_VECTOR(i, 1)
- FREQUENCY_VECTOR(i, 1)
- CALL_PRICE_VECTOR(i, 1)
- CALL_DATE_VECTOR(i, 1)
- MOODY_RATING_VECTOR(i, 1)
- SP_RATING_VECTOR(i, 1)
- INSURANCE_NAME_VECTOR(i, 1)
- CUSIP_VECTOR(i, 1)
- ISSUE_NAME_VECTOR(i, 1)
- BOND_TYPE_VECTOR(i, 1)

General Bond Information

After the data has been cleaned the code checks some conditions in the data it is looping through before continuing. If any of the conditions aren't met, then the code is sent to line 1983 and the next i is triggered. These conditions are:

VALOR_VECTOR, MATURITY_VECTOR, CURRENT_PRICE_VECTOR, PURCHASE_DATE_VECTOR, PURCHASE_PRICE_VECTOR, and PURCHASE_QUANTITY_VECTOR are not blank.

MATURITY_VECTOR, PURCHASE_DATE_VECTOR and PURCHASE_QUANTITY_VECTOR do not equal zero.

The function then starts inputting data from the vectors into their appropriate location in the output matrix TEMP0_MATRIX.

TEMP0_MATRIX(i, 1) is set equal to VALOR_VECTOR(i, 1)
TEMP0_MATRIX(i, 2) is set equal to MOODY_RATING_VECTOR(i, 1)
TEMP0_MATRIX(i, 3) is set equal to SP_RATING_VECTOR(i, 1)
TEMP0_MATRIX(i, 19) is set equal to CDate(MATURITY_VECTOR(i, 1))

The CDATE function changes the string to date format.

To place CURRENT_PRICE_VECTOR into TEMP0_MATRIX(i,61) two if-statements are used.

The first one checks to see if there is a "-"character in the current price. If there isn't then TEMP0_MATRIX(i, 61) is set equal to CURRENT_PRICE_VECTOR(i, 1).

Next there is a check to see if the value in TEMP0_MATRIX is greater than 0. If it is, then the value in TEMP0_MATRIX is converted to a double type variable.

If there is a "-" character in the CURRENT_PRICE_VECTOR value, then TEMP0_MATRIX(i, 61) is set equal to CURRENT_PRICE_VECTOR(i, 1) but no further conditions are tested.

The reasoning behind this has to do with the way bond prices can be formatted. Prices can be entered as a dollar figure and number of 32nds by using a "-"instead of a ".". For example a price of 101.25 could also be entered as 101-08.101-08, would represent 101 and 08/32 or 101.25. The code above ensures that if the price is not entered in the 32nds format the data is converted to a double variable to hold the decimal points.

Purchase Information

General Price Information Part I

TEMP0_MATRIX(i, 25) is set equal to CDate(PURCHASE_DATE_VECTOR(i, 1))

Again, the CDATE function changes the string to date format.

To place the PURCHASE_PRICE_VECTOR into TEMP_MATRIX(i, 26) the same logic that was applied to the CURRENT_PRICE_VECTOR is used again and for the same reasons.

TEMP0_MATRIX(i, 27) is set equal to CDbl(PURCHASE_QUANTITY_VECTOR(i, 1))

The CDbl function converts the value into a double variable and assumes the number of bonds purchased have a $1000 par value per bond.

For setting the PURCHASE_FEES_VECTOR equal to TEMP0_MATRIX(i, 27) an if-statement is used. This if-statement checks that the value of PURCHASE_FEES_VECTOR is greater than 0. If it is then TEMP0_MATRIX(i, 28) is set equal to CDbl(PURCHASE_FEES_VECTOR(i, 1)). If not then TEMP0_MATRIX(i, 28) is set equal to zero.

For the purchase fees it is important to note that they should not be re-entered if the fee is already included in the price. Also accrued interest should not be included as a fee.

COUPON_VECTOR and FREQUENCY_VECTOR are passed into their respective columns TEMP0_MATRIX(i,16) and TEMP0_MATRIX(i,17) in the same manner.

Both use an if-statement to check if the value being passed is greater than zero. If it is then the value is first converted to a double variable and then passed into the respective matrix.

If the value is less than zero then the value placed in TEMP0_MATRIX is zero.

CALL_PRICE_VECTOR and CALL_DATE_VECTOR are also passed to their respective matrixes TEMP0_MATRIX (i,23) and TEMP0_MATRIX(i,24) in similar ways.

For CALL_PRICE_VECTOR if the value being passed equals zero and for CALL_DATE_VECTOR if the value being passed is blank then the value placed in TEMP0_MATRIX for each respective column will be a blank space. If not then for CALL_PRICE_VECTOR the value will be converted to a double variable before being placed in TEMP0_MATRIX and for CALL_DATE_VECTOR the value is converted to date format before being placed in TEMP0_MATRIX.

Converting Price Data

The next section of code is responsible for taking the data from the Current Price column and Purchase Price column and converting the prices in those columns into one consistence format.

As discussed earlier in the section, price information can be entered in multiple formats. A further explanation is provided below.

Prices are entered as a percentage of par, which is assumed to be $1000. For example, if a $1000 dollar par bond is selling at a discount of $975 then the price is entered as $97.50.

The function, however, supports different pricing notations for quoting a bond. A user may input a bond using any of the methods described below:

Dollar Figure notation: Most familiar notation would quote the price like any normal price e.g. $97.50.

32^{nds} notation: Uses a "-" instead of a "." E.g. 101-08 is the same as 101 and 8/32 or $101.25.

The + notation: quoted as 99-10+ indicating to add an additional 1/16.

The 2 notation: quoted as 99-102 with the 2 indicating to add an additional 2/256.

In order to create a list of prices that are all consistent, the function uses the BOND_PRICE_PARSE_LINE to convert all of the possible notations into the Dollar Figure Notation. Converting the prices into this notation is also necessary in order to perform calculations with the prices later in the function.

To do this TEMP0_MATRIX (i,61) which was already set equal to the Current Prices is set equal to TEMP1_VAL. TEMP1_VAL is then sent through the sub BOND_PRICE_PARSE_LINE.

BOND_PRICE_PARSE_LINE returns an output with all the Current Prices now converted to the Dollar Figure Notation. The output with these prices is stored in the variable TEMP2_VAL. The new Adjusted Current Prices in dollar figure notation are then set equal to TEMP0_MATRIX(i,62).

The same process is then done with TEMP_MATRIX(i,26) which was already set equal to the Purchase Prices. These prices are all adjusted into Dollar Figure Notation using the sub BOND_PRICE_PARSE_LINE and the output is set equal to TEMP0_MATRIX(i,29).

TEMP0_MATRIX(i, 32) is then set equal to TEMP0_MATRIX(i, 27).

TEMP0_MATRIX(i, 33) is then set equal to TEMP0_MATRIX(i, 28).

TEMP0_MATRIX(i,74) which is the Market Value is set equal to the adjusted price of the bond multiplied by MULT1_VAL, multiplied by the adjusted quantity. The use of MULT1_VAL is to account for the fact that prices are entered as a percentage of par.

At this point the function also calculates the total Market Value of the portfolio by summing each individual bond's market value.

Credit Agencies Information

Moody's

This portion begins by setting the variable FIND_STR and TEMP0_MATRIX(i,10) equal to nothing.

An if-statement then begins to check that the ith value in TEMP0_MATRIX(i,2) does not equal the NR_STR variable recall that NR_STR equals "NR".

If it doesn't and if the CREDIT_AGENCY_FLAG is activated, then the LOOK_STR variable is set equal to the value in TEMP0_MATRIX(i,2). The value in TEMP0_MATRIX(i,2) recall was already set equal to the ith bond's Moody rating. The code is then sent to the sub CREDIT_RATINGS_MOODY_LINE.

The CREDIT_RATINGS_MOODY_LINE works by using the CREDIT_AGENCIES_MATRIX and LOOK_STR. The code loops through the CREDIT_AGENCIES_MATRIX until it reaches the rating that matches the one in LOOK_STR. When the rating is found the numeric position of the rating is stored in variable n and the sub returns to the main code.

When the code returns it checks to see if n is greater than 0. If it is then j is set equal to n and TEMP0_MATRIX(i, 6) which is the Moody Value of the bond is set equal to TEMP0_MATRIX(i, 74) which was already set equal to the market price of the bond. The function then also calculates the portfolio's Moody Value by summing the Moody Value of each bond.

At this point the reader might find this process redundant with the Market Value column discussed previously. The key difference between the Moody Value and the Market Value of a bond is that the Moody Value is only calculated when the bond has a Moody Rating. In other words all bonds will have a Market Value but not all bonds will have a Moody Value. This is also why the conditions for setting the Moody Value depend on n being greater than 0.

TEMP0_MATRIX(i,8) (the Moody's Rank column) and TEMP0_MATRIX(i,10) (the Average Rank Column) are then set equal to variable j.

S&P

Next a similar process then begins doing essentially the same operation that was done with the Moody Ratings for S&P Ratings.

First FIND_STR is set equal to nothing. Then the value in TEMP0_MATRIX(i,3), which holds the S&P ratings, is checked to make sure it does not equal NR_STR or in other words, it is checked to make sure it has a rating.

Then the code checks to see if the CREDIT_AGENCIES_FLAG is activated. If it is, then LOOK_STR is set equal to the value in TEMP0_MATRIX(i,3). The code is then sent to sub CREDIT_RATINGS_SP_LINE. This sub performs the same function to CREDIT_RATINGS_MOODY_LINE and outputs the rating in variable n. When the code returns, it checks that n is greater than zero. If it is, then j is set equal to n.

Similar to the Moody's section TEMP0_MATRIX(i,7) is set equal to TEMP0_MATRIX(i,74). Similar to above TEMP0_MATRIX(i,74) represents the Market value of a bond, TEMP0_MATRIX(i,7) is the S&P value of the bond. These two columns will equal each other only if the bond has an S&P rating.

The total Portfolio S&P Value however, is calculated differently than what was done with the Moody Value above. This time in order for the S&P value to be added to the portfolio the bond must have both

a Moody's and S&P rating. In the code this is expressed in the condition that TEMP0_MATRIX(i,6) <>"" in order for the ith bond to be included in the S&P portfolio Value.

TEMP0_MATRIX(i,9) (the S&P Rank column) is then set equal to j and

TEMP0_MATRIX(i,10) (the Average Rank column) is set equal to the value being currently stored from the Moody's section + the S&P ranking + tolerance all divided by 2. This value is then rounded to the nearest whole number.

Average Tier

The last part of the Credit Rating section is to determine the average tier of the bond. This is done using the average ranking in a process almost identical to the ones seen above.

First FIND_STR is set equal to nothing. Then an if-statement checks to see if there is an Average Rating value in the ith column of TEMP0_MATRIX(i,10) that is being passed. If there is and the CREDIT_AGENCIES_FLAG is activated then LOOK_STR is set equal to the value in TEMP0_MATRIX(i,10) and that value is converted into a string variable. LOOK_STR is then passed through CREDIT_RATINGS_TIER_LINE.

Like the two subs before it CREDIT_RATINGS_TIER_LINE uses the LOOK_STR and searches through the CREDIT_AGENCIES_MATRIX until it reaches the index value that matches the one in LOOK_STR. Once it finds the location, it then cross references the row number in the fourth column of the CREDIT_AGENCIES_MATRIX to find the Tier value that corresponds with the index number. The tier value is then returned in the variable n.

If n is greater than zero then j is equal to n and TEMP0_MATRIX(i,11) (the AVG Tier column) is set equal to j. If n is not greater than zero then TEMP0_MATRIX(i,11) is set equal to NR_INT.

If TEMP0_MATRIX(i,10) does not equal a value and the CREDIT_AGENCIES_FLAG is activated then LOOK_STR is set equal to NR_STR and is sent into the CREDIT_RATINGS_SP_LINE. When it returns if n is greater than zero then j is set equal to n and the average rating becomes the value in j.

If the CREDIT_AGENCIES_FLAG is not activated and TEMP0_MATRIX does not equal any value then the average rating is left blank and the Average Tier value will be set equal to NR_INT or 8.

General Price Information Part II

TEMP0_MATRIX(i, 12) is set equal to INSURANCE_NAME_VECTOR(i, 1)
TEMP0_MATRIX(i, 13) is set equal to CUSIP_VECTOR(i, 1)
TEMP0_MATRIX(i, 14) is set equal to ISSUE_NAME_VECTOR(i, 1)
TEMP0_MATRIX(i, 15) is set equal to BOND_TYPE_VECTOR(i, 1)

Next the code inputs the adjusted coupon/year into TEMP0_MATRIX(i, 18).

First if TEMP0_MATRIX(i, 17) (the coupon/ year which is inputted as Frequency) equals 0 then the adjusted coupon/year equals 1.

If the coupon/year equals 12 then the adjusted coupon/year equals 4.

For any other coupon/year the two are equal.

Next the function sets the value in COUNT_BASIS_VECTOR, which defaults to zero, equal to the variable B_VAL.

TEMP0_MATRIX(i, 20) which is the column for the first coupon month is then set equal to the month of the maturity date in TEMP0_MATRIX(i, 19).

TEMP0_MATRIX(i, 21) which is the years to maturity column then employs the YEARFRAC_FUNC using the SETTLEMENT date, maturity date and B_VAL to determine the value.

At the same time the portfolio line for the year to maturity column is calculated. It will report the year of the bond with the longest years to maturity. It does this by comparing each ith bond's year to maturity with the most recent bond that had the longest year to maturity. It uses the MAXIMUM_FUNC function to determine which one of the two numbers is larger. It then rounds the value.

TEMP0_MATRIX(i, 30) is set equal to the total cost of the bond and the total portfolio cost is calculated. This is done by multiplying the adjusted quantity by the adjusted price times MULT1_VAL and then adding the total product to the adjusted fees.

TEMP0_MATRIX(i,31) is set equal to the annual coupon and the total portfolio annual coupon is calculated. This is done by multiplying the adjusted quantity by MULT3_VAL and then multiplying the product by the coupon divided by MULT2_VAL.

To determine TEMP0_MATRIX(i,35) (Coupon Amount) an if statement is used. If the Coupon/Year is zero then so is the Coupon Amount. Otherwise it is equal to the Annual Coupon divided by Coupon/Year.

Taxes

The taxes sections begin by checking to see if the PORT_BOND_TYPES_FLAG is activated. If it is then LOOK_STR is set equal to TEMP0_MATRIX(i, 15) or the type of Bond. The LOOK_STR is then sent into the sub PORT_TAXES_LINE.

The PORT_TAXES_LINE works in a similar fashion to the rating subs. It uses the PORT_BOND_TYPES_MATRIX and LOOK_STR to find the tax amount that corresponds to the type of bond. When it is found it is set equal n equal to the Bond Type index and the tax percentage equal to FIND_STR.

If FIND_STR equals a value then TEMP0_MATRIX(i, 69) (the Bond Type Index column) is set equal to n otherwise it is set equal to nothing.

If the PORT_BOND_TYPES_FLAG isn't active then FIND_STR is set equal to nothing as well.

Monthly Cash Flows

To calculate the monthly cash flows from bond coupon payments a loop is used.

The loop, loops for j equals 1 to k where k=12 or the number of months.

Variable l is set equal to the Coupon/Year and variable m is set equal to the 1st coupon month.

If there is no Coupon/Year then l=0 and then TEMP0_MATRIX(i, 36 + j - 1) and TEMP0_MATRIX(i, 48 + j - 1) both equal 0. Or each month receives zero pre-tax and after-tax cash flow for that bond.

The reason for the j-1 in both the column section of the TEMP0_MATRIX is because the columns for the Pre-Tax payments and After-Tax payments start in column 36 and 48 respectively. In other words column 36 would be January's pre-tax cash flows, 37 would be February's pre-tax cash flows and so on. Same for the after tax cash flows. 48 would be January's after-tax cash flows the February in column 49 and so on.

If l doesn't equal 0 then TEMP1_VAL is set equal to (m - j) / (12 / l) it then uses TEMP1_VAL in the equation TEMP1_VAL-1 *Int(TEMP1_VAL/l). If the result of the last equation is zero then the month j has a coupon payment and TEMP0_MATRIX(i, 36 + j - 1) will be set to TEMP0_MATRIX(i, 35) (the Coupon Amount) otherwise it is set to zero.

To see why the above formula works, take for example a bond that pays cash 2 times a year with the first payment in January.

On the first pass j=1, l=2 and m=1. TEMP1_VAL=(1-1)/(12/2) = 0

0 – 1*int(0/1) = 0

The next coupon payment, which is in July, will be the only other time the formula equals zero.

.j=7, 1=2 and m=1. TEMP1_VAL =(1-7)/(12/2)=-1

-1-1*int(-1/1)=-1-1*-1 = -1 +1 = 0

At the same time the pre-tax portfolio cash flows is being calculated by adding all the coupon payments from each bond in the given month.

For the after tax cash flows the function first checks that FIND_STR is set to a value. If a value is being placed in the pre-tax cash flows section then the after tax of the cash flow is also determined using (1-FIND_STR). If a zero is placed in any month in the pre-tax section then the after-tax month will also be zero.

Yield to Maturity

Purchase Price

Yield to maturity based on purchase price is stored in TEMP0_MATRIX(i, 60) and is found using the PORT_BOND_YIELD_LINE using the following inputs:

P_VAL = Adj. Price + (Adj. Fees / MULT1_VAL) / Adj. Quantity)
S_VAL = Purchase Date
M_VAL = Maturity
C_VAL = Coupon / MULT2_VAL
F_VAL = Adj. Coupon/Year
R_VAL = 100
G_VAL = 0.5

Current Price

The next yield is the yield to maturity (YTM) based on the current price. This information is stored in TEMP0_MATRIX(i,62) and is also found using PORT_BOND_YIELD_LINE and the following changes to the inputs above:

P_VAL = Adj. Price
S_VAL = SETTLEMENT

Yield to Call

The next yield is the yield to call (YTC). First an if-statement is used to check three conditions. The first is that the bond has a call date and price and the last is that the call date is after the settlement date.

If all the conditions are met then the Yield to Call is found and stored in TEMP0_MATRIX(i, 64). Again the yield is found using the PORT_BOND_YIELD_LINE and the following changes to the inputs above:

P_VAL = Adj. Price
S_VAL = SETTLEMENT
M_VAL = Call Date
R_VAL = Call Price

Once the yield to call is found the code then checks to see which yield is lower between the YTM and YTC. The lower of the two yield is then placed in TEMP0_MATRIX(i, 65) with the corresponding maturity date being placed in TEMP0_MATRIX(i, 68). E.g. if the YTC is lower, then that yield will be used in the adjusted yield, and the call date will be used as the Adjusted maturity date.

If there are no call options on the bonds then the regular YTM and maturity date will be placed into TEMP0_MATRIX(i, 65) and TEMP0_MATRIX(i, 68) respectively.

TEMP0_MATRIX(i, 76) (the Adj. Lower Yield) is then set equal to TEMP0_MATRIX(i, 65).

Fair Yield to Maturity

This section uses current Market yield data to determine the YTM that would be paid on a similar grade bond with the same maturity.

First TEMP0_VAL is set equal to the number of years to maturity.

If that value is less than zero or greater than 29, TEMP0_MATRIX(i, 70) (the Fair YTM) is set equal to the Lowest Yield in column TEMP0_MATRIX(i, 65).

Otherwise, if the MARKET_YIELDS_FLAG and PORT_BOND_TYPES_FLAG are both activated and TEMP0_MATRIX(i, 69) (the Bond Type Index) has a value then the code precedes. If not the Fair YTM is set equal to the value in the Lowest Yield column.

The code then sets j equal to the value in Bond Type Index. LOOK_STR then finds the name of the bond type using the index number and the PORT_BOND_TYPES_MATRIX and converts the name to a string.

LOOK_STR is then sent to sub MARKET_YIELDS1_LINE where LOOK_STR is looped through the column headings until it finds the name in the PORT_BOND_TYPES_MATRIX that matches the one in LOOK_STR. The position number is then returned in the variable n.

If n is greater than zero then the sub MARKET_YILDS2_LINE is activated.

This sub will loop through the PORT_BOND_TYPES_MATRIX starting with the maturity column. It first looks to see if the value number in the maturity column its looping through is less than or equal to the Years to maturity of the bond being evaluated. It also looks to see if the number in the maturity column one position ahead is greater than the years to maturity of the bond being evaluated. If both conditions

are met then, if the position in the loop plus 2 is still less than the size of the data set (i.e. you are still three positions back from the last data point in the data set) then TEMP1_VAL is set equal to the yield value in column n that is one position ahead and TEMP2_VAL is set equal to the yield value in column n that is two positions ahead. If you are not three positions back from the last data point in the data set then TEMP1_VAL is set equal to the yield value in column n in the position it is currently in and TEMP2_VAL is set equal to the yield value in column n that is one position ahead.

If TEMP1_VAL and TEMP2_VAL do not come back blank then TEMP0_MATRIX(i, 70) is set equal to TEMP1_VAL + (TEMP2_VAL - TEMP1_VAL) * (TEMP0_VAL - ASYM_DOWN_FUNC(TEMP0_VAL, 1)). Note that the ASYM_DOWN_FUNC is a function similar to Int() which asymmetrically rounds numbers down.

If TEMP1_VAL and TEMP2_VAL do come back blank then TEMP0_MATRIX(i, 70) is just set equal to TEMP0_MATRIX(i, 69).

Next the function determines the Fair Price of the bond using the PORT_BOND_PRICE_LINE sub. To find the fair price the function uses the following inputs:

S_VAL = SETTLEMENT
M_VAL = Maturity
C_VAL = Coupon / MULT2_VAL
Y_VAL = Fair YTM
F_VAL = Adjusted Coupon/Year
R_VAL = 100

The output price is stored in TEMP0_MATRIX(i, 71).

The yield premium that the bond offers compared to bonds with a similar maturity and rating is then calculated by subtracting the Adjusted Lower Yield by the Fair YTM. The value is stored in TEMP0_MATRIX(i,72).

Value by Maturity Bracket

This section of the code categorizes each bond by their maturity bracket and places the Market Value of the bond into their appropriate bracket.

The code begins by setting variable k equal to h(1), which is the number of rows in the PORT_BOND_MATURITIES_MATRIX. A loop then begins from j = 1 to k.

First DATE1_VAL is set equal to the jth maturity date in PORT_BOND_MATURITIES_MATRIX.

If j is equal to 1 then the code checks to see if the maturity of the bond is less than or equal the first maturity date in PORT_BOND_MATURITIES_MATRIX. If it is, then the Market Value of the bond is placed into TEMP0_MATRIX(i, 76 + j). Since the Value by Maturity Brackets columns start at TEMP0_MATRIX(i, 77) and run for j number of columns, the 76 + j ensures the market value is placed correctly. In the portfolio line the sum of all the bonds market values with common maturities are calculated and placed in TEMP0_MATRIX(NROWS + 1, 76 + j).

If j is not equal to 1, then the code checks to see is the maturity of the bond is less than or equal to DATE1_VAL and greater than DATE2_VAL. To understand where DATE2_VAL comes from we must back

up a bit. In the first pass when j=1 the code will skip the "Else" condition that we are currently discussing. The last line before the loop loops is to set DATE2_VAL equal to DATE1_VAL. The "Else" situation that we are currently discussing can only occur after the first pass when j does not equal 1. So in this case the DATE1_VAL would be the jth maturity in PORT_BOND_MATURITIES_MATRIX and DATE2_VAL would be the jth-1 maturity in PORT_BOND_MATURITIES_MATRIX.

If these conditions are met, then the Market Value of the bond is placed into TEMP0_MATRIX(i, 76 + j) and again the portfolio line for the respective maturity date is calculated.

Value by Bond Type

The function also does something similar with the type of bond as it does with the maturity of the bond. This section categorizes the bonds by Type and places the Market Value of the bonds into their respective category.

First jj is set equal to hh + h(1). Recall that hh is equal to the minimum amount of cells outputted (76) and h(1) is equal to the number of rows in PORT_BOND_MATURITIES_MATRIX.

kk is then set equal to h(2) which recall is the number of rows in PORT_BOND_TYPES_MATRIX.

A loop then begins from j = 1 to kk.
If the type of bond being passed through the i loop is equal to the jth bond in PORT_BOND_TYPES_MATRIX then TEMP0_MATRIX(i, jj + j) is set equal to the Market Value of the bond. In the portfolio line the sum of all the bonds market values that are the same type of bond are calculated and placed in TEMP0_MATRIX(NROWS + 1, jj + j).

If the reader is confused by the logic behind jj+j in TEMP0_MATRIX, it helps to look at the final output.

Column 76	Column hh +1	Column hh + 2	…	Column hh + h(1)	Column jj + 1	Column jj + 2	…	Column jj + h(2)
V) ADJUSTED LY	**VI) VALUE BY MATURITY BRACKET: 1.0 Years**	**VI) VALUE BY MATURITY BRACKET: 2.0 Years**		**VI) VALUE BY MATURITY BRACKET: n Years**	**VII) VALUE BY BOND TYPE: A**	**VII) VALUE BY BOND TYPE: B**		**VII) VALUE BY BOND TYPE: n**

Since the number of columns that need to be filled and the column number that the values are place into are dependent upon the optional variables being present and the number of data points in the optional variables, this notation allows the function to be dynamic.

Value by Credit Quality

The last categorizing that the data does with the bonds and their respective Market Value is for Credit Quality. This process is almost identical to the one seen in Value by Bond type and proceeds as follows.

The function begins by setting jj equal to itself plus h(2). The reasoning for this should be clear given the example above. We want to start at the column right of jj+h(2), so our starting position is set to jj+h(2).

k is then set equal to h(3) and a loop begins from j= 1 to k.

If the value in TEMP0_MATRIX(i, 11) (the Average Tier), equals the j value being passed then the Market Value is placed into cell TEMP0_MATRIX(i, jj + j). At the same time the portfolio market value for each Credit Quality column is calculated by summing all the market values being placed into the respective column.

TEMP0_MATRIX(i, nn) is then set equal to interest earned. Recall that nn is the starting position of the last section. This is done by sending the purchase and settlement date through the YEARFRAC_FUNC function to return the year as a fraction. This is then multiplied by the adjusted quantity times MULT3_VAL. Then all this is multiplied by the coupon divided by MULT2_VAL.

As this is happening the portfolio line for the interest earned column is found by summing all the value being passed through TEMP0_MATRIX(i, nn).

The Total Return is then calculated by adding the market value to the interest earned, less the total cost all divided by the total cost.

Bond Analysis

This section contains a few analytics columns that show modified duration, bond risk, and bond sensitivity. Together, these metrics can be used to gauge the portfolio's sensitivity to changes in interest rates. In particular, the bond risk column attempts to estimate the change in market value in each bond and the portfolio overall that would likely result from an X basis point (Y%) change in interest rates.

The Modified Duration and Bond Sensitivity columns are used to give an indication of what might happen to the value of each holding and the portfolio overall as interest rates change.

– Notes from BOND_PORT_ANALYSIS_FUNC

The entire Bond analysis section is controlled by one if-statement. This if statement initially checks that there is a value in the YTM column TEMP0_MATRIX(i, 63) if there is then the code precedes.

If YTM is also greater than zero then, TEMP0_MATRIX(i, nn + 2) is set equal to modified duration. This is done using the BOND_CONVEXITY_DURATION_FUNC and the inputs:

S_VAL = SETTLEMENT
M_VAL = Maturity
C_VAL = Coupon / MULT2_VAL
Y_VAL = YTM
F_VAL = Adjusted Coupon/Year
R_VAL = 100

Next the function computes the change in price that would result in a 100 basis point decrease or increase in the bond. This information is stored in TEMP0_MATRIX(i, nn + 3) and TEMP0_MATRIX(i, nn + 4) respectively.

To do this, the code first checks that the YTM is greater than the DELTA_VAL. If it is, then Y_VAL is set equal to the YTM minus the DELTA_VAL, otherwise Y_VAL is set equal to epsilon.

The sub PORT_BOND_PRICE_LINE is used to determine the price.

For the increase in rates adjusted price the Y_VAL is set equal to YTM plus DELTA_VAL and PORT_BOND_PRICE_LINE is used again.

If the YTM is less than zero then,
TEMP0_MATRIX(i, nn + 2) is set equal to zero.
TEMP0_MATRIX(i, nn + 3) and TEMP0_MATRIX(i, nn + 4) are set equal to the current price.

Duration is then calculated and set equal to TEMP0_MATRIX(i, nn + 5). This is done by subtracting the Price if there is a drop in 100 basis points from the Price if there is a gain in 100 basis points. Divided by the product of 2 times the adjusted price times the DELTA_VALUE times (one plus YTM).

Next, if TEMP0_MATRIX(i, nn + 2) equals zero, then TEMP0_MATRIX(i, nn + 7) is set equal to zero. Otherwise the bond convexity is calculated. This is done by adding the Price if there is a gain in 100 basis points from the Price if there is a drop in 100 basis points minus the adjusted price. That value is then divided by the adjusted price times DELTA_VAL squared.

Next, TEMP0_MATRIX(i, nn + 9) is set equal to the Bond Risk which is a more accurate indicator of interest rate sensitivity that takes into account the effects of curvature in the relationship between price and yield. This is done by multiplying the modified duration by the DELTA_VAL and then subtracting that value by the product of the bond convexity times DELTA_VAL squared.

Finally, TEMP0_MATRIX(i, nn + 11) is set equal to Bond Sensitivity which is how much the value of a bond will change from a 100 basis points change in bond yield. This is done by multiplying the bond risk value by the market value of the bond.

The next i is then triggered and the whole giant i loop repeats.

Completing the Function

The Portfolio Lines

Once all the main inputs are placed in the matrix and the i loop finishes, the code begins to fill in any missing values; Most notably the values in the portfolio line.

This section begins with mm being set equal to jj. Recall that at this point jj is equal to hh + h(1) +h(2).

The value in TEMP0_MATRIX(NROWS + 1, 21) then has its decimals removed and a one added to it.

Next TEMP0_MATRIX(NROWS + 1, nn + 1) is set equal to (TEMP0_MATRIX(NROWS + 1, 74) + TEMP0_MATRIX(NROWS + 1, nn) - TEMP0_MATRIX(NROWS + 1, 30)) / TEMP0_MATRIX(NROWS + 1, 30).

TEMP0_MATRIX(NROWS + 1, 4) and TEMP0_MATRIX(NROWS + 1, 5) are set equal to zero.

Then the following columns also have their portfolio lines also set equal to zero:

Column 22, 34, 66, 67, 73, 75, nn +6, nn +8 and nn +10.

Next the variable ii is set equal to h(0) + h(1) + h(2) + h(3), jj is set equal to ii + h(2) + h(1), kk is set equal to jj + h(2) + h(3), ll is set equal to kk + h(2) and k is set equal to h(2)

A loop then beings for j = 1 to k to place zeros in the respective columns below:
TEMP0_MATRIX(NROWS + 1, ii + j)
TEMP0_MATRIX(NROWS + 1, jj + j)
TEMP0_MATRIX(NROWS + 1, kk + j)
TEMP0_MATRIX(NROWS + 1, ll + j)

Next ii is set equal to ii + h(2), jj is set equal to ll + h(2), kk is set equal to jj + h(1) and k is set equal to h(1)

A j loop begins again from 1 to k to place zeros in the following columns:
TEMP0_MATRIX(NROWS + 1, ii + j)
TEMP0_MATRIX(NROWS + 1, jj + j)
TEMP0_MATRIX(NROWS + 1, kk + j)

Finally, ii is set equal to ii + h(1) +h(2), jj is set equal to kk +h(1), kk is set equal to jj + h(3) and k is set equal to h(3).

A j loop begins again from 1 to k to place zeros in the following columns:
TEMP0_MATRIX(NROWS + 1, ii + j)
TEMP0_MATRIX(NROWS + 1, jj + j)
TEMP0_MATRIX(NROWS + 1, kk + j)

jj is then set equal to mm + h(3).

Second i Loop

An i loop then beings for i = 1 to NROW. This i loop is to complete some sections of the output that were not entered in the original i loop. These sections were not originally entered because they are dependent on some of the other outputs being completed first.

The first if-statement checks that there is a value in both the Moody Value and Moody Ranking column. If there is and if the Moody Value does not equal zero, then the Moody score is calculated by dividing the Moody value by the product of the Moody Value and Moody Ranking. The score is stored in TEMP0_MATRIX(NROWS + 1, 4). During this process the sum of the Moody Scores is also determined and stored in the Portfolio line in the Moody Score column.

The exact same process then ensures for the S&P Score, only using the S&P Value and S&P Ranking Column. The score is stored in TEMP0_MATRIX(NROWS + 1, 5).

Next the code checks to see if there is a value present in the Market Value column. If there is and if it doesn't equal zero then the percentage of that bond in the overall portfolio is calculated; This is done by dividing the Market Value of the bond by the Market value of the portfolio. The value is stored in TEMP0_MATRIX(i, 75). The sum of all the percentages is also calculated and stored in the Portfolio line in the Percent Portfolio column.

Weighted Columns

Next the Weighted Year to Maturity is calculated and the value is stored in TEMP0_MATRIX(NROWS + 1, 22). To do this, the function first checks that there is a value in both the Years to Maturity and Percent Portfolio columns. If there are then the two values are multiplied together and stored. At the same time the sum of all the weighted year to maturities is also calculated and stored in the Portfolio line in the Weighted Year to Maturity column.

Next the Weighted YTM is calculated and stored in TEMP0_MATRIX(i, 34). This is done by first checking that there are values in the Total Cost, and YTM at Buy columns, and also that the portfolio line under the Total Cost column has a value. If the conditions are met, then the Weighted YTM is calculated by multiplying the YTM at Buy by the Total Cost of the bond divided by the Total cost of the portfolio. Again, the Portfolio Weighted YTM is also calculated by summing the weights of each individual bond.

Next, the Premium Weight is calculated and stored in TEMP0_MATRIX(i, 73). To do this the function first checks that there is a value present in both the Percent Portfolio and Lower Yield Premium columns. If there is, then both values are multiplied together and the product is stored. The sum of each bond's Premium Weighting is also calculated for the Portfolio line.

Next the code checks to see if the PORT_BOND_TYPES_FLAG is activated. If it is, then LOOK_STR is set equal to the Type of bond and sent to the sub PORT_TAXES_LINE. The sub returns with the tax rate that corresponds to that type of bond stored in FIND_STR. If the PORT_BOND_TYPES_FLAG is not activated then FIND_STR is set to blank.

Next if there are values in the Percent Portfolio and Adjusted Lower Yield columns then the weighted yield is determined by multiplying the two values together. Once again the Portfolio line is also determined by summing all the weighted yield values.

The code then checks to see if there is a value being stored in FIND_STR. If there is, then the weighted yield is multiplied by 1 – the FIND_STR. This figure represents the after tax weighted yield and is placed in TEMP0_MATRIX(i, 67). Again, the sum of all the weighted after tax yields is calculated and stored in the portfolio line.

Next, the Duration Weight column is calculated and stored in TEMP0_MATRIX(i, nn + 6). The process is similar to the ones above. First the Percent Portfolio and Modified Duration columns are checked to make sure there is a value in them and if there is they are multiplied together. The portfolio line is then calculated the same as for the other columns above.

The Convexity Weight column is calculated in a similar manner as the Duration Weight column. Only the Bond Convexity and Market Value columns are used. After they are checked to make sure they both have values, the total market value of the portfolio is checked to make sure it also has a value. If it also does then TEMP0_MATRIX(i, nn + 8) is set equal to the Market Value of the Bond divided by the Market Value of the portfolio multiplied by the Bond Convexity. The total portfolio Convexity Weight is also determined in the same fashion as above.

Finally, the Risk Weight column is calculated and stored in TEMP0_MATRIX(i, nn + 10). This is done first by checking that there are values in the Percent Portfolio and Bond Risk columns. If there are, then the two values are multiplied together and stored. The total portfolio Risk Weight is also determined in the same fashion as above.

Next kk is set equal to jj + h(2), ll is set equal to kk + h(1) + h(2) * 3 + h(3), mm is set equal to ll +h(1) and k is set equal to h(1)

Completing Sections for the Optional Values

A loop then beings for j = 1 to k

The first if-statement is for finding the values for Credit Quality by Bond Type section. This value is found by multiplying the Average Ranking by the value in the ii + jth column of Value By Bond Type and dividing that by the Portfolio's Value By Bond Type of the ii + jth column. Like the if-statements above, this calculation doesn't proceed unless all of the values needed to calculate it are present. The Portfolio line is then also determined by summing each value in the Values for Credit Quality by Bond ii+jth column.

The next if-statement is for determining the Modified Duration by Bond Type section. The logic for the code is exactly the same as determining the values for the Credit Quality by Bond Type section, only this time the input Average Ranking is substituted for Modified Duration. The rest of the if-statement logic is identical. The values, however, are stored in the kk + jth column in the code.

The next if-statement is for determining the Pre-Tax Yield by Bond Type and store the values in the ll + jth column. This is done using the same equation above with the only change to the inputs being that the Adjusted Lower Yield is used in place of the Average Ranking. The Portfolio line for this section is also determined using the same methods as discussed above.
This if-statement has an added condition that if FIND_STR equals a value then the function also inputs the After Tax Yield by Bond Type. It does this by multiplying the value in the ll +jth column by 1 minus

the appropriate tax rate. The information is stored in the mm +jth column. The sum of all the values in each column is also calculated for the Portfolio Line.

The next four if-statements follow almost identical logic to the if-statements seen above.

The first two are used to find the information for the Credit Quality by Maturity Bracket section and the information for the Pre-Tax and After-Tax Yield by Maturity Bracket sections. These are most similar to the first and third if-statements seen for the Credit Quality sections discussed directly above.

The last two are used to find the information for the Modified Duration by Credit Quality and the Pre-Tax and After-Tax Yield by Credit Quality sections. These are most similar to the second and third if-statements seen for the Credit Quality sections discussed directly above.

The next i is then triggered and the loop repeats with the next i value.

More Portfolio Lines

TEMP0_MATRIX(NROWS + 1, 4) is set equal to Round(TEMP0_MATRIX(NROWS + 1, 4), 2)
TEMP0_MATRIX(NROWS + 1, 5) is set equal to Round(TEMP0_MATRIX(NROWS + 1, 5), 2)
TEMP0_MATRIX(NROWS + 1, 13) is set equal to (TEMP0_MATRIX(NROWS + 1, 4) +
TEMP0_MATRIX(NROWS + 1, 5)) / 2
TEMP0_MATRIX(NROWS + 1, 14) is set equal to "Avg Rating"
TEMP0_MATRIX(NROWS + 1, 16) is set equal to TEMP0_MATRIX(NROWS + 1, 22)
TEMP0_MATRIX(NROWS + 1, 17) is set equal to "Avg Yrs Maturity"
TEMP0_MATRIX(NROWS + 1, 60) is set equal to TEMP0_MATRIX(NROWS + 1, 34)
TEMP0_MATRIX(NROWS + 1, 63) is set equal to TEMP0_MATRIX(NROWS + 1, 66)
TEMP0_MATRIX(NROWS + 1, 65) is set equal to TEMP0_MATRIX(NROWS + 1, 66)
TEMP0_MATRIX(NROWS + 1, 68) is set equal to "Avg Yield"
TEMP0_MATRIX(NROWS + 1, 72) is set equal to TEMP0_MATRIX(NROWS + 1, 73)
TEMP0_MATRIX(NROWS + 1, nn + 2) is set equal to TEMP0_MATRIX(NROWS + 1, nn + 6)
TEMP0_MATRIX(NROWS + 1, nn + 7) is set equal to TEMP0_MATRIX(NROWS + 1, nn + 8)
TEMP0_MATRIX(NROWS + 1, nn + 9) is set equal to TEMP0_MATRIX(NROWS + 1, nn + 10)
TEMP0_MATRIX(NROWS + 1, nn + 11) is set equal to TEMP0_MATRIX(NROWS + 1, 74) *
TEMP0_MATRIX(NROWS + 1, nn + 10)

CREDIT_AGENCIES_FLAG section

The last three if statements all check to see if CREDIT_AGENCIES_FLAG is active.

The first if-statement uses the CREDIT_AGENCIES_LINE sub and the Portfolio's overall Moody's score. The sub returns a text rating based on the weighted average Moody's rating for the overall portfolio. This value is stored in TEMP0_MATRIX(NROWS + 1, 2).

The next if statement does the same thing only with the S&P score. The text rating is stored in TEMP0_MATRIX(NROWS + 1, 3).

The last if-statement uses a loop to provide a text rating for every column in the Credit Quality by Bond Type section of the output (Section 9). This is done in the same manner as above. In all cases if CREDIT_AGENCIES_FLAG is not active the output is left blank.

Select Case Output

The function can produce six different output depending on what the user specifies.

They are summarized in the table below

Case	Output
0	TEMP0_MATRIX
1	TEMP1_MATRIX- Portfolio Cash Flow Table
2	TEMP2_MATRIX- Bond Portfolio Maturity Table
3	TEMP3_MATRIX- Portfolio Bond Type Table
4	TEMP4_MATRIX- Portfolio Credit Quality Table
Not Specified	All possible outputs