YIELD_SENSITIVITY_FUNC TECHNICAL REPORT

RAFAEL NICOLAS FERMIN COTA

Abstract

This Technical Report provides a full explanation of the YIELD_SENSITIVITY_FUNC function, as well as explanations for the supporting functions within it.

Table of Contents

Introduction to the YEILD_SENSITIVITY_FUNC function

About the function

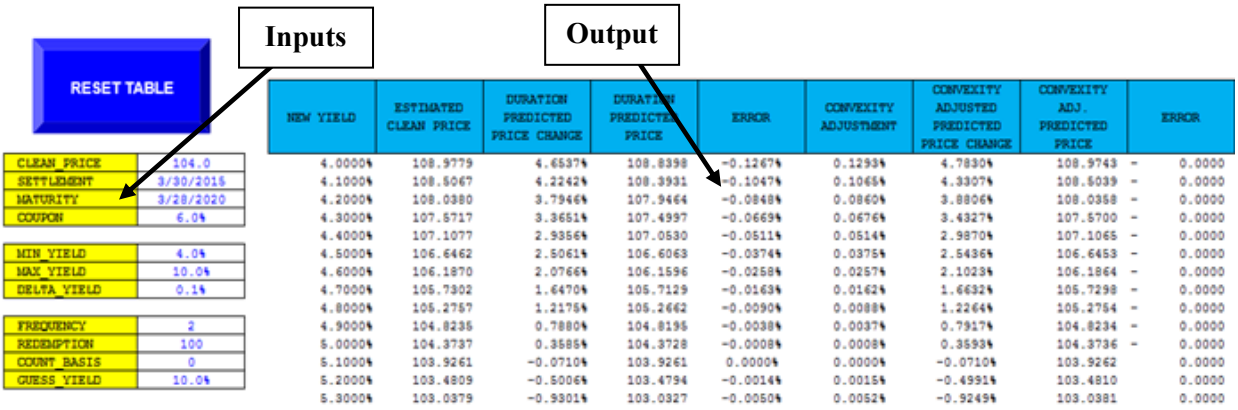
The purpose of the YIELD_SENSITIVITY_FUNC function is the generate bond prices given a range of specified yields. As the name of the function suggests it provides sensitivity analysis on bond price movements relative to changes in interest rates.

The function utilizes the popular bond portfolio management measurements for risk, duration and convexity, in order to provide a robust analysis for its user.

The function is capable of producing two outputs depending on what the user specifies.

For a simple analysis of a bond the user may simply request the yield to maturity, along with the modified duration and convexity. This output, however, will not generate any sensitivity to yield changes. For the remainder of the document this output will be known as *Output One.*

For a more thorough analysis the user may request the second output. This output provides the estimated clean price, the duration predicted price change (percentage), the duration predicted price, the percentage error between the duration predicted price change and the estimated clean price, the convexity adjustment, the convexity adjusted predicted price change (percentage), the convexity adjusted predicted price, and the percentage error between the convexity adjusted predicted price and the estimated clean price; for each new yield. This second output is also the default output used in the TEST_VALIDATION workbook and is printed to the CONVEX worksheet. For the remainder of the document this output will be known as *Output Two.* An example of the excel worksheet is provided below.



It is important to note that to produce the different outputs, the function calls multiple functions within the given code. As such, the YIELD_SENSITIVITY_FUNC function can be thought of as an amalgamation of several functions used to determine all of the required outputs. Because these secondary functions play such an important role in the YIELD_SENSITIVITY_FUNC function, they will also be discussed in the content of this technical report for the benefit of the reader.

Understanding Duration and Convexity

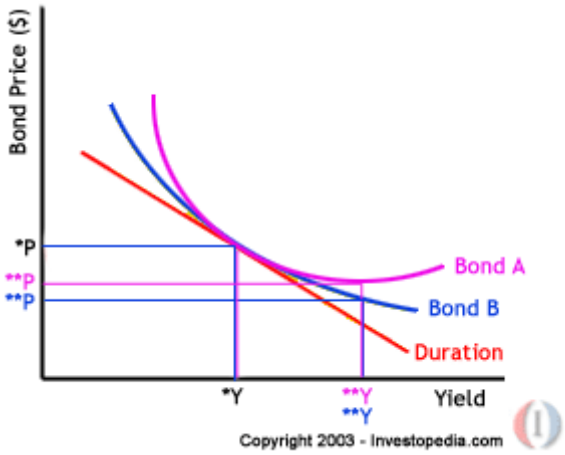
In order to gain a full understanding of the YIELD_SENSITIVITY_FUNC function it is important for the user to first appreciate the concepts that underscore its purpose.

In bond portfolio management Duration and Convexity can be used as a measure of risk when assessing a particular bond's merits.

Duration in the context of the bond asset class provides two valuable measurements for an investor. First, it is a measure of how long, in years, it takes for the price of a bond to be repaid. And second, and more specific to the function, it acts as a measure of risk by assessing a bond's price volatility relative to an increase or decrease in interest rates.

While Duration can provide a rough estimate in the price of a bond given a specific change in interest rates it alone would not be enough. The price – yield relationship of a bond is convex in nature, however, the Duration function is linear. This means that in extreme cases when interest rates fall or rise, Duration will underestimate the upside and overestimate the downside. In order to compensate for this investors look for a convexity adjustment to better estimate prices changes in a bond for a given change in interest rates.

Convexity is also a useful measure when assessing bonds that have the same duration. If for instance bond A and bond B have the same duration but bond A is more convex that bond B, then bond A may be more favorable because of its ability to better capture upside and limit downside. Refer to the diagram below.



Setting-up the Function

Function Variables

Input Variables

Variable	Type	Description
Output	Integer	Determines which of the two outputs the function will produce. If this input equals zero, the function will produce Output Two; if this input equals any other integer besides zero, the function will produce Output One.
Clean_Price	Double	Input for the current clean price (cash price minus accrued interest) of the bond.
Settlement	Date	The date the transaction would settle (the date you take possession of the bond).
Maturity	Date	The date the bond matures.
Coupon	Double	The annual coupon rate for the bond.
Min_Yield	Double	The minimum yield that the array will evaluate. Sets the value for the first entry under the “New Yield” column (Column E of the Spreadsheet).
Max_Yield	Double	The maximum yield that the array will evaluate. Sets the value for the last entry under the “New Yield” column.
Delta_Yield	Double	Sets the increments in which the yield increases by in the “New Yield” column. As a result, Delta_Yield will determine the number of entries in the “New Yield” column and also, the size of the array.
Frequency	Integer; Optional	The frequency with which the coupons are paid. Default convention is semi-annual coupon payments (Value of 2).
Redemption	Double; Optional	Par value of the bond, or amount of principal that is returned at the maturity date. Default setting is redemption which equals 100.
Count_Basis	Integer; Optional	The day count convention that the bond uses. The default value is 0.
Guess_Yield	Double; Optional	Defines the initial interval of uncertainty for the optimization performed by the Parab_Zero_Func. The default value is 10%.

Counter Variable

Variable	Type	Description
i	Long	Used in loop to place required data in appropriate rows for Output Two.

Boundary Variable

Variable	Type	Description
NROWS	Long	Used to identify the number of rows that are required for the final matrix of Output Two.

Holder Variables

Variable	Type	Description
PYIELD_VAL	Double	Holds output from the BOND_YIELD_FUNC function, used in Output One.
CONVEXITY_VAL	Double	Holds the value for convexity, used in Output One.
MDURATION_VAL	Double	Holds the value for the modified duration, used in Output One.

Matrix Variable

Variable	Type	Description
TEMP_MATRIX	Variable	Used to hold the output from the BOND_CONVEXITY_DURATION_FUNC function early in the code. Then reused to hold the final output for both Output One and Output Two.

Error Handling

Once the function's inputs and variables have been declared, the function begins with some initial error handling.

First the minimum yield is checked to make sure that it is less than the maximum yield. Next the delta yield is checked to make sure that it does not equal zero. If any of these condition are not met then the function is sent to the ERROR_LABEL and the function returns the error number.

This should intuitively make sense since the minimum yield required for sensitivity purposes cannot equal or be greater than the maximum yield in the analysis and the delta yield, which represents the incremental increase in yield in the sensitivity analysis, cannot equal zero.

Output One

Determining Values for Output One

Yield to Maturity

The first value that is found for output one is the percent yield or yield to maturity. This is done using the BOND_YIELD_FUNC function and the output is set equal to the variable PYIELD_VAL. While an explanation of the BOND_YIELD_FUNC function is given in the Secondary Functions section a brief discussion is provided below.

The formula for determining the YTM of a bond is:

$$P_0^b = \sum_{t=1}^{M} \frac{CF_t}{(1+y)^t}$$

Equation 1: Source Fidelity Investments

Where P is the present value price of a bond, CF are the constant cash flows and y is the discount factor or in this case the YTM.

In a scenario where all the cash flows from a bond are consistent (coupon and principal are equal) y can be solved for by simply rearranging the formula. But in most cases when the cash flows from a bond are not equal, an iterative process is needed. This iterative process tries different y values (YTM values) until the above equation equals the current price of the bond.

The BOND_YIELD_FUNC function completes this task using the parabola zero solver function to iterate YTM until it finds a YTM that produces a price for a bond equal to the current clean price.

Convexity and Modified Duration

Next the convexity and modified duration of the bond is determined. This is done using the BOND_CONVEXITY_DURATION_FUNC function. This secondary function is also discussed more in-depth in the Secondary Functions section, and for now it is only important to note that the function produces four outputs that are placed in TEMP_MATRIX. These four outputs are convexity, modified duration, duration and the bond cash price.

For the purposes of Output One only the first two are needed.

These two values are extracted out of TEMP_MATRIX and placed in the holder variables CONVEXITY_VAL and MDURATION_VAL respectively.

Completing the Function for Output One

Once the values for output one are determined, the only thing left to do is format the data into a matrix.

TEMP_MATRIX is once again used but this time to house the final output for Output One. TEMP_MATRIX is re-dimensioned to have three rows and two columns.

In the first three rows in the first column the headings are placed into the matrix. They are, PERCENT YIELD, MODIFIED DURATION and CONVEXITY.

Next the values for the respective headings are placed into the matrix: PYIELD_VAL, MDURATION_VAL and CONVEXITY_VAL.

The function is then set equal to TEMP_MATRIX and the function ends.

Output Two

Placing the Headings

If the second output option is chosen, the code will still Determine the values used in Output One as some will still be needed for Output Two, however will bypass the code that was discussed in: *Completing the Function for Output One.*

In this case the function moves to the next section of code for Output Two.

The code begins by determining the number of rows that will be needed in for the final output matrix. This is found by subtracting the max yield figure from the min yield figure and dividing by the delta yield.

This number is then set equal to the variable NROWS.

TEMP_MATRIX, which will again be used to house the final output is re-dimensioned to have NROWS plus 1 number of rows and nine columns. The actual number of rows in TEMP_MATRIX will actually be one more than NROWS plus one, since it is dimensioned in base 0. This is done in order to place the headings in the first row while still keeping the rest of the code in base 1 format.

If the reader is wondering why a one is added to NROWS to determine the number of rows it is because NROWS, calculated the way it was, will always be one row short the actual number of rows needed. Consider the formula to determine the number of values in a dataset:

(MAX VALUE – MIN VALUE)/ Incremental increase (decrease) + 1

Consider the following case in the data set 1, 1.5, 2, 2.5, 3. The max number = 3, the min number = 1 and the incremental increase = 0.5. Using the formula initially used to calculate NROWS, we would determine that there are 4 values in the data set. However that would be incorrect, there are 5. The plus one used in re-dimensioning TEMP_MATRIX accounts for this kind of discrepancy.

Next the headings for the output are placed into the first row of the matrix in each of the nine columns. In order they are: NEW YIELD, ESTIMATED CLEAN PRICE, DURATION PREDICTED PRICE CHANGE, DURATION PREDICTED PRICE, ERROR, CONVEXITY ADJUSTEMENT, CONVEXITY ADJUSTED PREDICTED PRICE CHANGE, CONVEXITY ADJ PRIDICTED PRICE, and ERROR.

In first row of the first column, under the heading NEW YIELD the MIN_YIELD value is placed.

Determining and Formatting the Values

In the code for Output One, the values were first calculated and then in a different processed passed into TEMP_MATRIX. In Output Two, however, this process occurs simultaneously.

Because Output Two produces sensitivity analysis on the factors identified in the headings for given changes in yield, the calculations for each heading must be performed repeatedly for each new yield.

To do this a loop is used.

This loop loops from 1 until NROWS + 1. Remember, NROWS+1 represents all the yields in the data set.

The if Statement

This simple if-statement is used for placed the NEW YIELD values into their respective cell.

For every i other than the first one, the NEW YIELD value that is placed in the first column of the ith row is equal to the value in the previous column plus DELTA_YIELD (the incremental increase).

For the first i, the first row of the first column directly under the heading NEW YIELD the MIN_YIELD value is placed.

Placing the Values

The i-loop continues after the if-statement, and for each value of i the following process ensues to place all the values into their respective place within TEMP_MATRIX:

Heading/ Output	Row	Column	Function/ Formula Used	Inputs
ESTIMATED CLEAN PRICE	i	2	BOND_CASH_PRICE_FUNC	SETTLEMENT, MATURITY, COUPON, TEMP_MATRIX(i, 1), FREQUENCY, REDEMPTION, COUNT_BASIS, 1
DURATION PREDICTED PRICE CHANGE	i	3	DELTA_DURATION_PRICE_FUNC	MDURATION_VAL, TEMP_MATRIX(i, 1), PYIELD_VAL
DURATION PREDICTED PRICE	i	4	CLEAN_PRICE * (1 + TEMP_MATRIX(i, 3))	N/A
ERROR	i	5	(TEMP_MATRIX(i, 4) - TEMP_MATRIX(i, 2)) / TEMP_MATRIX(i, 2)	N/A
CONVEXITY ADJUSTEMENT	i	6	0.5 * CONVEXITY_VAL * ((TEMP_MATRIX(i, 1) - PYIELD_VAL) ^ 2)	N/A
CONVEXITY ADJUSTED PREDICTED PRICE CHANGE	i	7	DELTA_CONVEXITY_PRICE_FUNC	CONVEXITY_VAL, MDURATION_VAL, TEMP_MATRIX(i, 1), PYIELD_VAL
CONVEXITY ADJ PRIDICTED PRICE	i	8	CLEAN_PRICE * (1 + TEMP_MATRIX(i, 7))	N/A
ERROR	i	9	(TEMP_MATRIX(i, 8) - TEMP_MATRIX(i, 2)) / TEMP_MATRIX(i, 2)	N/A

Further detail on the functions used can be found in the Secondary Functions Section. Furthermore, the formulas used are financial formals for calculating Convexity and Duration adjustments and are given below.

DURATION PREDICTED PRICE = Original Bond Price * [1 + Duration Predicted Price Change (%)]

CONVEXITY ADJUSTEMENT = 0.5 * Convexity Value * change in YTM

CONVEXITY ADJ PRIDICTED PRICE = Original Bond Price * [1+ Convexity Adjusted Predicted Price Change (%)]

The function is then set equal to TEMP_MATRIX and the function ends.

Secondary Functions

Introduction to the Secondary Functions

This section provides an explanation to the support functions used within the YEILD_SENSITIVITY_FUNC function. The explanation to these functions were authored by Zoubin Irani.

BOND_YIELD_FUNC

Purpose:

The BOND_YIELD_FUNC utilizes PARAB_ZERO_FUNC and CALL_BOND_YIELD_OBJ, along with the bond's clean price, settlement date, maturity date, coupon rate, frequency of coupon payments, redemption value, day count convention, and guess yield as inputs, to accurately calculate the yield to maturity of a bond.

Inputs:

Input	Required?	Description
Clean_Price	Yes	The value of the bond less accrued coupon.
Settlement	Yes	The date that the bond is active (either the day the trade is executed or a few days after).
Maturity	Yes	The date that the bond expires and the initial face value is returned.
Coupon	Yes	The interest percent earned on the initial face value of the bond.
Frequency	Optional	The number of interest (coupon) payments made to the bond holder per annum. The default value is 2 (semi-annual).
Redemption	Optional	The price at which the issuing company can repurchase a bond, prior to maturity. Default is 100.
Count_Basis	Optional	The number of days between two coupon dates as decided by market convention (Ex. convention could dictate that each month has 30 days and that a year has 360 days). Default is 0.
Guess_Yield	Optional	A starting point for the "guess and check" process of calculating bond yield. Default is 0.3.

Further Explanation

The function calculates yield to maturity by utilizing PARAB_ZERO_FUNC (which is explained separately).

Within PARAB_ZERO_FUNC, CALL_BOND_YIELD_OBJ_FUNC will be used as an input, and is the function that calculates yield to maturity. However, CALL_BOND_YIELD_OBJ_FUNC needs to be iterated several times to converge to a correct yield to maturity, and needs to be analyzed to see if there were no errors in the calculation.

Therefore, BOND_YIELD_FUNC's purpose is to utilize CALL_BOND_YIELD_OBJ_FUNC, PARAB_ZERO_FUNC, and other inputs to arrive at an accurate yield to maturity.

Through the use of PARAB_ZERO_FUNC, CALL_BOND_YIELD_OBJ_FUNC will be looped (utilized) 600 times, and each time it is used, it should get closer and closer to the correct yield to maturity. After 600 times, CONVERG_VAL, which is affected by PARAB_ZERO_FUNC, is checked to see if it is within the specified tolerance level. If it is not within the tolerance level, then CALL_BOND_YIELD_OBJ_FUNC did not converge to one solution as planned, and the GUESS_YIELD is used. Likewise, if PARAB_ZERO_FUNC returns a yield greater than 2 ^ 52, the yield is thrown out as a calculation error resulted in a yield far too big, and the GUESS_YIELD is used. Otherwise, the calculated yield is used, which is the ideal solution.

PARAB_ZERO_FUNC

Purpose:

The purpose of the function is to find the local minimum of a function through vertical parabola interpolation. This function is used in BOND_YIELD_FUNC to help converge on an accurate yield to maturity of a bond, given the inputs.

Inputs:

Input	Required?	Description
Lower_Val	Yes	PARAB_ZERO_FUNC requires an interval of uncertainty to start the iterations. Lower_Val is the lower bound of the uncertainty interval.
Upper_Val	Yes	PARAB_ZERO_FUNC requires an interval of uncertainty to start the iterations. Upper_Val is the upper bound of the uncertainty interval.
Func_Name_Str	Yes	PARAB_ZERO_FUNC requires a function, f(x) that it will find a local minimum for. The function should be inputted as a string.
Converg_Val	Optional	The value that you want f(x*) to converge to. The default value is zero.
Counter	Optional	Counts the number of iterations that have elapsed. The default value is zero.
NLOOPS	Optional	Defines the number of iterations that the function should perform until it should exit the function (if the function has not found x* which satisfies the defined tolerance). The default value is 100.
Tolerance	Optional	The error range that the function is willing to accept. In many cases, a precise value for x*, where f(x*) = Converg_Val, will only be found when nLOOPS approaches infinity, therefore the tolerance allows for an acceptable difference between f(x*) and Converg_Val. The default value is 0.000000000000001.

Further Explanation:

For a given function, f(x), PARAB_ZERO_FUNC continuously shrinks the interval of uncertainty [Lower_Val, Upper_Val] until it finds a point within the interval that satisfies the specified tolerance level, or until the number of specified loops/iterations have elapsed. If the number of loops have elapsed without a satisfactory point being returned, the function will return 0 and reset the Converg_Val equal to 2.

Looking back at BOND_YIELD_FUNC, PARAB_ZERO_FUNC was used to determine a maturity rate. If the correct (or near-correct) yield to maturity rate is found, then the absolute difference between the Upper_Val and Lower_Val should be smaller than the tolerance rate, and the calculated yield will become the value of the PARAB_ZERO_FUNC.

CALL_BOND_YIELD_OBJ_FUNC

Purpose:

This function returns the price of the bond given its yield. It is used to call the BOND_CASH_PRICE_FUNC to pass a yield and return the bond price.

Inputs:

X_VAL: The yield input used to calculate the bond price.

Sub-Inputs:

The characteristics of the bond that will be priced with yield, X_VAL, and the actual clean price of the bond are pulled from the inputs prescribed in the BOND_YIELD_FUNC:

PUB_BOND_ARR(1): Coupon
PUB_BOND_ARR(2): Frequency
PUB_BOND_ARR(3): Redemption
PUB_BOND_ARR(4): Count_Basis
PUB_BOND_ARR(5): Clean_Price
PUB_BOND_ARR(6): Maturity
PUB_BOND_ARR(7): Settlement

Further Explanation:

The function calculates an error value by taking the difference between the bond price with the yield (X_VAL) and the actual bond price, and squaring it. The function provides an indication of how close the yield (X_VAL) is to the true yield to maturity. The function will be used in the optimization calculation, PARAB_ZERO_FUNC, to find the correct yield to maturity of a bond. When the difference between the bond price with the yield (X_VAL) and the actual bond price equals zero (or some tolerance that is very close to zero), then X_VAL is equal to the yield to maturity.

BOND CASH_PRICE_FUNC

Purpose:

The function will return the cash or clean price of a security that pays periodic interest. In addition, based on the OUTPUT input, the function will add or not add accumulated interest.

Inputs:

Input	Required?	Description
Maturity	Yes	When the bond is set to mature/expire, and the face value is returned.
Coupon	Yes	The interest percent earned on the initial face value of the bond.
Yield	Yes	The rate for discounting cash flows to determine the clean price (i.e. the Return on Investment).
Settlement	Yes	The settlement date (the date the bond trade settles).
Frequency	Optional	Frequency of interest coupon payments per annum. The default value is 2 (semi-annual coupon payments).
Redemption	Optional	The price at which the issuing company can repurchase a security (bond, in this case) prior to maturity. The default value is 100.
Count_Basis	Optional	The number of days between two coupon dates as decided by market convention (Ex. convention could dictate that each month has 30 days and that a year has 360 days). The default is Case 0. Case 0, 1, 4 represent 30 day months with 360 days in a year. Case 2 represents the actual days in a month with 360 days in a year. Case 3 represents the actual days in a month with 365 days in a year.
Output	Optional	Depending on the output, the BOND_CASH_PRICE_FUNC will add the accumulated interest to the price. If it is Case 0, the function returns the cash price with accumulated interest. If it is not equal to Case 0, it will return the clean price. The default value is Case 0.

Further Explanation:

The function discounts the future cash flows of the bond using k-compounding convention.

There are three possible cases.

If the settlement date comes after the maturity date, then the BOND_CASH_PRICE_FUNC = 0.

Next, if the settlement and the maturity date are the same then the BOND_CASH_PRICE_FUNC equals the redemption value for the bond (par value + coupon on the maturity date).

If those two cases are invalid, then the function calculates either the cash price or clean price of the bond, using the specified market convention.

Select Case COUNT_BASIS determines the month/year market convention used. In addition, Select Case Output decides whether to use the Cash Price (Case 0) or Clean Price (any Case besides 0) of the bond.

This function is used along with BOND_CASH_PRICE_FUNC to determine the price of the bond given its yield.

COUPPCD_FUNC

Purpose:

This function calculates the number of coupon payments that are remaining in the bond.

Inputs:

Input	Required?	Description
Settlement	Yes	The settlement date (the date the bond trade settles).
Maturity	Yes	When the bond is set to mature/expire, and the face value is returned.
Frequency	Optional	Frequency of interest coupon payments per annum. The default value is 2 (semi-annual coupon payments).

Further Explanation:

There are four possible cases that will evaluate how many coupon payments are remaining.

First, if there is less than one coupon payment per year, then the COUP_NUM=0, and therefore there are no remaining coupons.

Next, if the maturity date of the bond comes before the settlement date (date that the bond was purchased), then there are no coupon payments remaining. This would likely never happen, but it is included to make the model more adaptable.

Last, if the settlement date and the maturity date are on the same day (the bond is purchased on the same day it expires), then the bond owner can expect one more coupon payment, so COUP_NUM=1. This makes sense because when the maturity date arrives, the bond holder receives the last coupon payment as well as the principal amount.

If those three cases do not occur, then the EDATE_FUNC sets the date value to the settlement date and loops through, adding 6 months each time to the date. It loops until the date exceeds maturity date. Each loop adds one to the counter (j) and then, it returns j (which is set to equal the COUPNUM_FUNC, and therefore the number of coupons that are remaining in the bond.)

EDATE_FUNC

Purpose:

This function will return a serial number that is the indicated number of months before or after the specified start date, and a maturity date that falls on the same day of the given start date.

Inputs:

Input	Required?	Description
Date_Val	Optional	Date_Val is a date that represents the start date. To avoid problems, dates should be entered using the Excel DATE function. For example, use DATE(2008,5,23) for the 23rd day of May, 2008. Problems can occur if the dates are entered as text. The default value is 0.

Months	Optional	The number of months before or after the specified start date, wherein a positive equals a future date and a negative equals a past date. The default value is 1.

Further Explanation:

The beginning of the function states that if the Date_Val = 0, then the function should use the current date as the start date. Next, the function will use the Date_Val result and add/minus the specified number of months after the start date (based on the Months input) to give a maturity date.

This function is used within several other functions, including COUPNUM_FUNC, COUPPCD_FUNC, and COUPNCD_FUNC.

COUPDAYBS_FUNC

Purpose:

The function returns (not calculates) the number of days from the previous coupon payment to the settlement date.

Inputs:

Input	Required?	Description
Settlement	Yes	The settlement date (the date the bond trade settles).
Maturity	Yes	When the bond is set to mature/expire, and the face value is returned.
Frequency	Optional	Frequency of interest coupon payments per annum. The default value is 2 (semi-annual coupon payments).
Count_Basis	Optional	The number of days between two coupon dates as decided by market convention (Ex. convention could dictate that each month has 30 days and that a year has 360 days). The default is Case 0.

Further Explanation:

Date_Val is a variable which calls the COUPPCD_FUNC function, and uses the settlement date (Settlement), maturity of the bond (Maturity), and the number of interest payments made per year as inputs to determine the days from the previous coupon date until settlement date (Frequency).

Afterwards, “i", another input, through calling the COUNT_DAYS_FUNC function, uses the output from Date_Val, along with the settlement date (Settlement), and the number of days between two coupon dates (Count_Basis), to determine the number of days between the previous coupon payment.

It returns an error in two specific situations:

If Frequency (the number of interest payments to the bond) is less than 1.

If Maturity (the maturity date when the bond expires) is before Settlement (the settlement date).

COUPPCD_FUNC

Purpose:

This function will calculate (not return) the number of days from the previous coupon date until the settlement date.

Inputs:

Input	Required?	Description
Settlement	Yes	The settlement date (the date the bond trade settles).
Maturity	Yes	When the bond is set to mature/expire, and the face value is returned.
Frequency	Optional	Frequency of interest coupon payments per annum. The default value is 2 (semi-annual coupon payments).

Further Explanation:

There are three cases.

First, if the Frequency is less than 1, then COUPPCD_FUNC = 0 and therefore the number of days from the previous coupon date until the settlement date is 0.

Second, if Maturity <= Settlement, then COUPPCD = 0.

Third, using the COUPNCD_FUNC (which will represent the next coupon payment date), the COUPPCD_FUNC references the next coupon payment date and then uses the EDATE_FUNC to identify how many months earlier the last coupon payment was. Therefore, if there are semi-annual coupons, then the output will be 6 months previous from the next coupon date.

COUPNCD_FUNC

Purpose:

This function returns the next coupon payment date for a bond.

Inputs:

Input	Required?	Description
Settlement	Yes	The settlement date (the date the bond trade settles).
Maturity	Yes	When the bond is set to mature/expire, and the face value is returned.
Frequency	Optional	Frequency of interest coupon payments per annum. The default value is 2 (semi-annual coupon payments).

Further Explanation:

There are three cases.

First, when the Frequency is less than 1 (i.e. No coupon payments to the bond holder) the output is 0 because there is no next coupon payment date.

Second, if the maturity date is before the settlement date, the output is also 0 days because there are no more coupon payment dates. If the Maturity and Settlement date are the same, the function outputs the maturity date implying that the next coupon payment is the same date as the date of purchase.

Third, if the maturity date comes after the settlement date, the function identifies how many coupon payments are remaining less the final maturity payment. By referencing the EDATE_FUNC, COUPNCD_FUNC can use the Frequency input to determine how many months before the maturity date the next coupon payment is due.

COUNT_DAYS_FUNC

Purpose:

The function calculates the number of days between two specified dates.

Inputs:

Input	Required?	Description
Start_Date	Yes	The start date, needs to be a “date” type variable
End_Date	Yes	The end date, needs to be a “date” type variable
Count_Basis	Optional	The number of days between two coupon dates as decided by market convention (Ex. convention could dictate that each month has 30 days and that a year has 360 days). The default is Case 0. For a value of 0 COUNT_DAYS_FUNC applies a NASD approach to calculating days (30/360) For a value of 1,2,3 the COUNT_DAYS_FUNC calculates the exact number of days between the Start_Date and End_Date (actual convention approach) For a value of 4 or greater, COUNT_DAYS_FUNC applies a European approach to calculating days (Europe 30)

Further Explanation:

The function first assigns the start date and end date as date entities, then the day, month, and year as numerical values.

Afterwards, the function ensures that the start date and end date cannot be zero and that the end date cannot be before the start date (an error will be given otherwise).

Next, the function calculates the difference between the in the number of days between Start_Date and End_Date, based on the Count_Basis input.

COUPDAYSNC_FUNC

Purpose:

This function calculates the time to the next coupon payment in days.

Inputs:

Input	Required?	Description
Settlement	Yes	The settlement date (the date the bond trade settles).
Maturity	Yes	When the bond is set to mature/expire, and the face value is returned.
Frequency	Optional	Frequency of interest coupon payments per annum. The default value is 2 (semi-annual coupon payments).
Count_Basis	Optional	The number of days between two coupon dates as decided by market convention (Ex. convention could dictate that each month has 30 days and that a year has 360 days). The default is Case 0. For a value of 0, COUNT_DAYS_FUNC, the function called within COUPDAYSNC_FUNC, applies a NASD approach to calculating days (30/360) For a value of 1,2,3 the COUNT_DAYS_FUNC, the function called within COUPDAYSNC_FUNC, calculates the exact number of days between the Start_Date and End_Date (actual convention approach) For a value of 4 or greater, COUNT_DAYS_FUNC, the function called within COUPDAYSNC_FUNC, applies a European approach to calculating days (Europe 30)

Further Explanation:

There are two cases.

The first case is, if there are no coupon payments left (frequency < 1) or the maturity date is before the settlement date, then the output would be zero.

The second case is, the function counts the days between now and the next coupon payment date by setting DATE_VAL (which calls on COUPNCD_FUNC), as the next coupon payment date. Afterwards, “i”, a variable, calls on COUNT_DAYS_FUNC to count the number of days between the Settlement date and DATE_VAL, based on the Count_Basis input.

ACCRINT_FUNC

Purpose:

Returns the accrued interest of a security (bond) that pays periodic interest since its last coupon payment.

Inputs:

Input	Required?	Description
Settlement	Yes	The settlement date (the date the bond trade settles).
Maturity	Yes	When the bond is set to mature/expire, and the face value is returned.
Coupon	Yes	The interest percent earned on the initial face value of the bond.
Frequency	Optional	Frequency of interest coupon payments per annum. The default value is 2 (semi-annual coupon payments).
Count_Basis	Optional	The number of days between two coupon dates as decided by market convention (Ex. convention could dictate that each month has 30 days and that a year has 360 days). The default is Case 0. For a Case value of 0 or 4, US (NASD) 30/360 is used, which means that each month has 30 days and there are 360 days in a year. In Case 1, the real dates are used (actual month lengths and actual year lengths). In Case 2, the actual month lengths are used but years are only 360 days. In Case 3, the actual month lengths are used but years are always 365 days.

Further Explanation:

First, the function checks if the maturity date is equal to the settlement date or before the settlement date. As well, the function checks if the frequency of interest payments is less than 1 per annum or if there is a coupon rate of zero or less. If any of these are true then the accrued interest is set to zero and the function exits, as there is a problem with one of the inputs.

Afterwards,

- Pdays_Val, calling on the COUPDAYBS_FUNC, calculates how many days since the last bond payment.
- Ndays_Val calling on the COUPDAYSNC_FUNC, calculates how many days until your next bond payment.
- Variable “J” is Pdays_Val plus Ndays_Val, which represents the time that has accrued.

If no time has accrued, J would equal 0, and therefore ACCRINT_FUNC will also be set to 0 as no interest would have accrued.

Next, based on the case selected from Count_Basis, the amount of interest is calculated by using Coupon, Frequency, and Factor_Val (which is the variable "J" after it goes through Count_Basis) as inputs.

BOND_CONVEXITY_DURATION_FUNC

Purpose:

The function calculates and returns the modified and Macaulay duration and Convexity Table of a bond. These values will be used as a measure of a bond price's response to changes in yield.

Inputs:

Input	Required?	Description
Settlement	Yes	The settlement date (the date the bond trade settles).
Maturity	Yes	When the bond is set to mature/expire, and the face value is returned.
Coupon	Yes	The interest percent earned on the initial face value of the bond.
Yield	Yes	The rate for discounting cash flows to determine the clean price (i.e. the Return on Investment).
Frequency	Optional	Frequency of interest coupon payments per annum. The default value is 2 (semi-annual coupon payments).
Redemption	Optional	Par value of the bond, or the amount of principal that is returned at the maturity date. Default setting is 100.
Count_Basis	Optional	The number of days between two coupon dates as decided by market convention (Ex. convention could dictate that each month has 30 days and that a year has 360 days). The default is Case 0.
Output	Optional	If it is Case 0, the function returns the cash price with accumulated interest. If it is not equal to Case 0, the function will return the clean price. The default value is Case 0.

Further Explanation:

First, the function checks if the maturity date is less than the settlement date, in which case the function returns zero as the maturity date should never be less than the settlement date.

Afterwards, the function calls the BOND_DATES_BOND_TENOR_FUNC, using Settlement, Maturity, Frequency, and Count_Basis as inputs, to create a vector of payment periods in terms of years from purchase date.

Next, there are two possible scenarios based on the Output variable.

If Output variable is 0, then an array listing the convexity, modified and Macaulay duration, and price of the bond characterized by the provided inputs will be produced.

These values are calculated through several steps. First, the function sums the present value of coupons (Temp1_sum) using a loop, which also discounts each coupon payment with their respective yield and discount period. Afterwards, the duration (Temp2_sum) is calculated by using a loop that sums the time

weighted PV of each coupon. Then, the convexity is calculated (Temp3_Sum) by using a loop that sums the convexity values. From these sums, BOND_CONVEXITY_DURATION_FUNC will output an array with the convexity as the first element, modified duration as the second element, Macaulay duration as the third element, and the bond price as the fourth element.

However, if the output variable is not 0, then the output will be a "i x 7" (i representing number of payments) column table with each column representing Tenor, Payments, Discount Factors, PV of Payments, PV Weights, Duration, and Convexity. These values are calculated in a similar fashion as the method used when the Output variable is 0.

BOND_DATES_BOND_TENOR_FUNC

Purpose:

The purpose of BOND_DATES_BOND_TENOR_FUNC is to take the settlement date and the maturity date as an input, with optional inputs of frequency and count basis, in order to calculate the amount of time left until the bond matures. This is known as the tenor. In other words, the function creates a vector of payment periods in terms of years from the settlement date. The output is useful in determining bond duration and convexity as both relate to the timing of the payments.

Inputs:

Input	Required?	Description
Settlement	Yes	The settlement date (the date the bond trade settles).
Maturity	Yes	When the bond is set to mature/expire, and the face value is returned.
Frequency	Optional	Frequency of interest coupon payments per annum. The default value is 2 (semi-annual coupon payments).
Count_Basis	Optional	The number of days between two coupon dates as decided by market convention (Ex. convention could dictate that each month has 30 days and that a year has 360 days). The default is Case 0. Case 0, 4, and 1 use Actual Days Per Month, Actual Year Length to calculate the amount of time left until the time matures. Case 2 and 3 remove leap year factors by using exactly 360 or 365 days, respectfully, for the year length.

Further Explanation:

There are four cases.

First, if the frequency of the coupon payment is less than one, then the function equals zero (BOND_DATES_BOND_TENOR_FUNC = 0), meaning there is no tenor.

Second, if the settlement date is after the maturity date, the bond has already matured, and therefore the tenor is zero.

Third, if the settlement date equals the maturity date, then the tenor equals the maturity date.

If the first three cases do not occur, then the function will calculate Pdays_Val, Ndays_Val, and NSIZE.

Pdays_Val represents the number of days since the last coupon payment, calculated by calling COUPDAYBS_FUNC and using Settlement, Maturity, Frequency, and Count_Basis as input.

Ndays_Val represents the number of days until the next coupon payment, calculated by calling COUPDAYSNC_FUNC and using Settlement, Maturity, Frequency, and Count_Basis as input.

NSIZE is the addition of NDAYS_VAL and PDAYS_VAL, representing the time between two coupon payments.

Afterwards, based on the Case selected for Count_Basis, the Temp_Factor, which represents the proportion of time that has passed since the last coupon payment divided by the entire time between two coupon payments (fractional payments), will be calculated.

Next, variable "j" will be calculated, which is equal to the number of coupon payments left in the bond. "j" is calculated by calling on the COUPNUM_FUNC and having Settlement, Maturity, and Frequency as inputs.

Afterwards the Temp_Vector is set up.

In the Temp_Vector's first cell, the equation of (1/Frequency) – (Temp_Factor/Frequency) represents the time that has passed since settlement, with 1/Frequency representing the time remaining until the bond matures and Temp_Factor/Frequency representing the number of days between coupon payments divided by the number days in a year.

From cell 2 until the cell containing the last coupon payment, the previous cell's value is added to 1/Frequency (which again, represents the portion of time until the bond matures). This sum is the amount of time between each coupon payment. The sum of the sums will be equal to the amount of time until the bond matures.

Last, the function is this array/value with BOND_DATES_BOND_TENOR_FUNC = Temp_Vector.

DELTA_DURATION_PRICE_FUNC

Purpose:

The function calculates the change in a bond's price given a change in yield and the bond's duration.

Inputs:

Input	Required?	Description
MDuration_Val	Yes	The modified duration of the bond.
Yield1_Val	Yes	The new yield of the bond.
Yield0_Val	Yes	The original yield of the bond.

Further Explanation:

The function calculates the change in the bond price if the yield was to move from Yield0_Val to Yield1_Val.

Please note this function does not take into account a change in a bond price due to a change in convexity, which is dealt with through DELTA_CONVEXITY_PRICE_FUNC.

The formula for DELTA_DURATION_PRICE_FUNC is (–MDuration_Val * (Yield1_Val – Yield0_Val)).

Since modified duration is the first derivative of bond price with respect to yield, modified duration multiplied by the change in yield will approximate the percentage change in bond price. Since the bond pricing function is not linear, this calculation will only be accurate for very small changes in yield (when Yield1_Val is close to Yield0_Val).

DELTA_CONVEXITY_PRICE_FUNC

Purpose:

This function calculates the change in a bond's price given a change in the bond's yield, taking convexity into account.

Inputs:

Input	Required?	Description
Convexity_Val	Yes	The convexity of the bond.
MDuration_Val	Yes	The modified duration of the bond.
Yield1_Val	Yes	The new yield of the bond.
Yield0_Val	Yes	The original yield of the bond.

Further Explanation:

Since the relationship between bond price and yield is convex, duration changes when yield changes. To more precisely calculate the change in bond price for a specified change in yield (especially when Yield0_Val and Yield1_Val are not close; large change in yield) this function takes into consideration the incremental impact that convexity (the second derivative of bond price with respect to yield) has on the change in a bond's price.

Therefore, by taking convexity into account, Yield0_Val and Yield1_Val do not have to be close, in order to calculate an accurate change in the bond's price.

Convexity is taken to account through Adj_Val, which is (0.5 * Convexity_Val * ((Yield1_Val – Yield0_Val) ^ 2)). Adj_Val's formula approximates the incremental impact of convexity on the bond price.

The output of Adj_Val and the sum of DELTA_DURATION_PRICE_FUNC, with MDuration_Val, Yield1_Val, and Yield0_Val as inputs, then equals the DELTA_CONVEXITY_PRICE_FUNC.

Overall, the DELTA_CONVEXITY_PRICE_FUNC is a more precise approximation of the percentage change in bond price for the specified yield change, because it takes into consideration both duration