When creating financial models for company valuation, circular references always occur. Although the Excel built-in iteration system might generate the correct answer, a user-defined function is preferred as we can actively modify the function to improve its speed, accuracy and functionality. Further, a user-defined function can be used as part of a program that automates building financial models.

Arts and Crafts of PE Modelling

Solving Circular References

Mateusz Hojdysz, Yi Kang, Marissa MacLellan, Steven Walt

Contents

Motivation for Creating Circular Reference Solving Functions

With financial statement data accessible online, there are programs that aggregate data and perform data sanitization to generate historical financial statements. Given this fact, it is viable to build a program that automates financial modeling. While most calculations in financial models are simple arithmetic, there are several key circular references that require identification of the difference equation in order to find the equilibrium solution. Normally, the Excel built-in iteration system is used to solve for circular reference. It cannot be used once the calculations are conducted off worksheet. Therefore, a user-defined function for solving circular reference is required for financial modeling automation.

Coding a set of Circular Reference Solving Function

Driving philosophy of coding a function to solve circular reference

The goal of a circular reference solving function is to find the stable equilibrium solution of a difference function in the form of $x_{n+1} = f(x_n, c, v)$, where x represents the element that the function will solve, c is a vector of constants, and v is a vector of variables dependent on x_n and c.

Structure of a General Circular Reference Solving Function

There are three parts of each set of circular reference solving functions:

1. The main body
2. The objective function
3. The optimization function

The main body's purpose is to take parameters, initialize the optimization process, process singular calculations, and eventually return the equilibrium solution. The objective function is used to call the calculation process portion of the main body. The optimization function calculates the equilibrium solution by utilizing a numerical optimization method which will call the objective function as part of the iterative process.

Structure of a Main Body Function

Parameter Intake

There are three kinds of parameters required for the Main Body Function: constants required to perform one iterative process, the value of the sought after element as optional to determine whether the Main Body Function is called for the first time, and miscellaneous optional parameters to determine the output and optimization set-ups.

Calculation Process

When the main body function is called for the first time (i.e. when there is no value assigned to the sought after element), the function will proceed to a subroutine to load the constants into a private global array (the subroutine is usually called LOAD_LINE). Then, the function will proceed

into another subroutine (normally named SOLVE_LINE) to call the Optimization function which will return the equilibrium solution.

When the Main Body Function is called with a starting value for the sought after element, the Main Body Function will perform one calculation process and produce an ending value of the sought after element. When the absolute difference between the starting value and the ending value is less than an error term (normally less than 10^-4), the ending value is the equilibrium solution to the circular reference.

Structure of an Objective Function

Parameter Intake

The Objective Function only takes in a value assigned to the sought after element.

Calculation Process

The Objective Function will call the Main Body Function by passing the element value as the starting value of one calculation process.

Structure of an Optimization Function

Parameter Intake

The Optimization Function takes in the name of the Objective Function in string format, and the initial guessing values.

Calculation Process

The calculation process varies depending on the optimization method it utilizes; nevertheless, it always involves calling the Objective Function iteratively until there is convergence among the ending values of the sought after element.

Choice of Optimization Method

Normally, there are two families of optimization methods: the Newton family and the Bisection family. A member of the Newton family is the Muller's method, and a member for the Bisection family is the Nelder-Mead simplex method. The Newton family typically tries to estimate the equilibrium solution via polynomial curve fitting (area for 3-d, etc.) and has faster convergence speeds than the Bisection family. However, lack of convergence will occur if the epsilon is too small, too many calculation steps that increases the machine rounding error, or the function exhibits small variations with large input perturbation. The Bisection family searches the equilibrium solution through zoning. Therefore, once an equilibrium solution is located in a range, the optimization process will continuously narrow that range until the equilibrium solution is found; a method from the Newton's family may exhibit explosive behavior if the objective function is not structured 'nicely'.

In practice, Muller's method is always preferred if it can produce viable results with a small enough error term; otherwise, the Nelder-Mead simplex method should be utilized to solve for the equilibrium point.

Circular Reference Function for solving Implied Share Price (DCF)

Data Required

The required parameters are: number of option/warrant tranches, array of the option/warrant tranches with the first column as number of shares and the second column as exercise price, number of convertible security tranches, array of convertible security tranches with the first column as the tranche size and the second column as the conversion price, basic share outstanding, and the implied equity value.

Description of a Single Calculation Process

1. Calculate the total additional shares if all in the money options are exercised.
2. Calculate the total share repurchase based on exercised in the money options.
3. Calculate the number of additional share result from conversion or convertible securities
4. Total shares after dilution = basic shares outstanding + additional shares from option exercising + additional shares from convertible security conversion
5. Update the Implied Share Price $\text{Implied Share Price} = \frac{\text{Implied equity value}}{\text{Total share after dilution}}$

Choice of Optimization Method

Upon testing, Muller's method is viable with an error term of 10^-3, which is sufficient when calculating share price as the result is accurate at the cent level.

Circular Reference Function for solving Operating Lease (DCF)

Data Required

The required parameters are: an array of future operating lease commitments, current year and previous year of long term debt, short term debt, and capital lease obligation, current year interest expense, and effective interest rate from the previous year and two years ago.

Description of a Single Calculation Process

1. Aggregate the total present value of the operating lease commitments using a for loop
2. Calculate the operating lease depreciation $= \frac{\sum \text{PV of operating lease commitments}}{\text{Number of years of commitment}}$
3. Calculate the adjustment to operational earning
   $\text{Operational earning}_{\text{Adj}} = \text{PV of Operating Lease}_{\text{Next}} - \text{Operating lease depreciation}$
4. Update capital lease obligation
   $\text{Capital lease obligation}_{\text{Curr}} = \text{CLO}_{\text{Curr}} + \sum \text{PV of operating lease commitments}$
5. Calculate the updated current year of effective interest rate
   $\text{Effective interest rate}_{\text{Curr}} = \frac{\text{total interest payment}}{\text{total amount of debt outstanding}}$
6. (Optional) Update the cost of debt by averaging the three year trailing effective interest rate.

Choice of Optimization Method

Upon testing, Muller's method can produce an equilibrium solution with an error term of 10^-5, which is sufficient.

Circular Reference Function for solving Cash Balance (DCF)

Data Required

The parameter-intakes are segregated into three arrays – asset items, liability and equity items, net income items, and miscellaneous items – previous year total asset, effective tax rate, effective interest rate, LTD/last year assets, and money market interest rate.

Description of a Single Calculation Process

1. Calculate the value of specified asset – all asset items except cash
2. Calculate the value of long term debt based on current year and previous year total asset
3. Calculate the interest expense based on long term debt
4. Calculate the value of net income and addition to retained earnings, and then the updated value of retained earnings
5. Calculate the value of specified liabilities – all liability and shareholder's equity item except short term debt
6. Calculate the value of net required financing
7. Finally derive an updated value for cash based on the net required financing.

Choice of Optimization Methods

Upon testing, Muller's method can produce an equilibrium solution with an error term of 10^-5, which is sufficient.

Circular Reference Function for solving Net Income (LBO)

Data Required

The parameter-intakes are segregated into four arrays: non-debt and miscellaneous elements, revolver elements, term loan elements, non-term loan elements. Array is used instead of individual elements to maintain the total parameters intake under the limit of 30. The segregation of different type of debt related elements is to facilitate later calculations by determining the number of debt items in each category. This set is to ensure the flexibility of the function to accommodate changes of the debt structure.

Description of a Single Calculation Process

1. From a given value of Net Income, derive the drawdown/repayment of the revolver facility.
2. Calculate the optional payments to each term loan recursively based on the amount of cash leftover, with more description in the section below.
3. Calculate the total cash flow from financing activities, and therefore the cash balance.
4. Calculate interest income based on cash balance, interest expense based on revolver, term-loans, non-term loans balance, and commitment fee from unused revolver balance.
5. Finally derive an updated value for Net Income.

Calculating Optional Payments

The optional payments for each term loan are considered according to their rankings; for example, optional payment for subordinate debt is only considered if there is cash left over after paying the optional payment for senior debt. Therefore, recursive calculation is logically sound in this situation.

In the function OP_PAYMENT_RECUR_FUNC, the optional payment is calculated for the most senior debt first; the function is exited after all the optional payments are calculated, in the order of seniority. The number of recursive calling is adjusted based on the number of term-loans to ensure the flexibility of this model.

Choice of Optimization Methods

The Nelder-Mead simplex method is chosen to solve for convergence in this case. The NM method is chosen to combat convergence issue at a cost of convergence speed compare to techniques such as Newton's method or Muller's method. The $f_{Net\ Income}$ produces approximately 1000th of change in magnitude given a change in the input parameter, this phenomenon causes Newton's and similar method will experience convergence issue for an acceptable epsilon (around 10^4, since the unit used in the model is in millions of dollars); they will produce a guess near the correct answer, and the subsequent guess will diverge. The NM method, however, once identifies the area that contains the correct answer, will stay in that area until a satisfactory answer is identified. Therefore, the NM method is used to solve for Net Income instead of Muller's methods.

Appendix – VBA codes

Solving for Operating Lease

Main Body

```
Option Explicit
Option Base 1
Private PUB_SOLVE_OPLEASE_ARR As Variant
Function SOLVE_OPLEASE(ByRef OPLEASE_RNG As Variant, _
ByRef LTD_RNG As Variant, _
ByRef STD_RNG As Variant, _
ByRef CAPLEASE_RNG As Variant, _
ByRef INT_VAL As Variant, _
ByRef INT_RATE_PREV_RNG As Variant, _
Optional ByRef DEBT_COST_VAL As Variant, _
Optional ByRef OUTPUT As Integer = 0, _
Optional LOW_VAL As Double = 0.0001, _
Optional UP_VAL As Double = 0.5) As Double

'placeholder for cost of debt, and addition to Operation earnings
Dim X_VAL As Double
Dim Y_VAL As Double

'counter
Dim i, NYEAR As Integer

'declare array to store transferred data
Dim OPLEASE_ARR As Variant
Dim INT_RATE_PREV_VAL As Variant
Dim LTD_VAL As Variant
Dim STD_VAL As Variant
Dim CAPLEASE_VAL As Variant

OPLEASE_ARR = OPLEASE_RNG
INT_RATE_PREV_VAL = INT_RATE_PREV_RNG
LTD_VAL = LTD_RNG
STD_VAL = STD_RNG
CAPLEASE_VAL = CAPLEASE_RNG
```

Main Body Cont.

```
'check if the function is called by user or Muller
If IsMissing(DEBT_COST_VAL) Then
   GoSub LOAD_LINE:
   GoSub SOLVE_LINE:
   Exit Function
End If

'declare placeholders for calculation
Dim TOT_OPLEASE_PV As Double
Dim OPLEASE_DEP As Double
Dim ADJ_OP_EARNING As Double

'find out how many years of operating lease is there
NYEAR = UBound(OPLEASE_ARR, 1)

'find out the PV of the operating leases
TOT_OPLEASE_PV = 0
For i = 1 To NYEAR
   TOT_OPLEASE_PV = TOT_OPLEASE_PV + OPLEASE_ARR(i, 1) * (1 + DEBT_COST_VAL) ^ (-i)
Next i

'depreciation
OPLEASE_DEP = TOT_OPLEASE_PV / NYEAR
'adjustment to operational earning
ADJ_OP_EARNING = OPLEASE_ARR(1, 1) - OPLEASE_DEP
'adjust long term debt
CAPLEASE_VAL(1, 2) = CAPLEASE_VAL(1, 2) + TOT_OPLEASE_PV
'find out new cost of debt
X_VAL = INT_VAL / (LTD_VAL(1, 1) + LTD_VAL(1, 2) + STD_VAL(1, 1) + STD_VAL(1, 2) +
(CAPLEASE_VAL(1, 1) + CAPLEASE_VAL(1, 2)) / 3)
X_VAL = (X_VAL + INT_RATE_PREV_VAL(1, 1) + INT_RATE_PREV_VAL(1, 2)) / 3
SOLVE_OPLEASE = Abs(X_VAL - DEBT_COST_VAL)
Exit Function
```

Main Body Cont.

```
LOAD_LINE:
   ReDim PUB_SOLVE_OPLEASE_ARR(1 To 6, 1 To 1)
   PUB_SOLVE_OPLEASE_ARR(1, 1) = OPLEASE_ARR
   PUB_SOLVE_OPLEASE_ARR(2, 1) = LTD_VAL
   PUB_SOLVE_OPLEASE_ARR(3, 1) = STD_VAL
   PUB_SOLVE_OPLEASE_ARR(4, 1) = CAPLEASE_VAL
   PUB_SOLVE_OPLEASE_ARR(5, 1) = INT_VAL
   PUB_SOLVE_OPLEASE_ARR(6, 1) = INT_RATE_PREV_VAL
Return

SOLVE_LINE:
   X_VAL = MULLER_ZERO_FUNC(LOW_VAL, UP_VAL, "SOLVE_OPLEASE_OBJ")
   Dim OPLEASE_PV As Double
   OPLEASE_PV = PUB_SOLVE_OPLEASE_ARR(5, 1) / (X_VAL * 3 - PUB_SOLVE_OPLEASE_ARR(6, 1)(1, 1)
- PUB_SOLVE_OPLEASE_ARR(6, 1)(1, 2)) - PUB_SOLVE_OPLEASE_ARR(2, 1)(1, 1) -
PUB_SOLVE_OPLEASE_ARR(2, 1)(1, 2) - PUB_SOLVE_OPLEASE_ARR(3, 1)(1, 1) -
PUB_SOLVE_OPLEASE_ARR(3, 1)(1, 2) - (PUB_SOLVE_OPLEASE_ARR(4, 1)(1, 1) +
PUB_SOLVE_OPLEASE_ARR(4, 1)(1, 2)) / 3

   Select Case OUTPUT
   Case 0 'cost of debt
      SOLVE_OPLEASE = X_VAL
   Case 1 'adj to operating earning
      SOLVE_OPLEASE = PUB_SOLVE_OPLEASE_ARR(1, 1)(1, 1) - OPLEASE_PV /
UBound(PUB_SOLVE_OPLEASE_ARR(1, 1), 1)
   Case 2 'adj to amount of debt
      SOLVE_OPLEASE = OPLEASE_PV
   End Select
Return

End Function
```

Objective Function

```
Function SOLVE_OPLEASE_OBJ(ByVal X_VAL As Double) As Double

Dim Y_VAL As Double

Y_VAL = SOLVE_OPLEASE(PUB_SOLVE_OPLEASE_ARR(1, 1), _
PUB_SOLVE_OPLEASE_ARR(2, 1), _
PUB_SOLVE_OPLEASE_ARR(3, 1), _
PUB_SOLVE_OPLEASE_ARR(4, 1), _
PUB_SOLVE_OPLEASE_ARR(5, 1), _
PUB_SOLVE_OPLEASE_ARR(6, 1), _
X_VAL)

SOLVE_OPLEASE_OBJ = Y_VAL

End Function
```

Solving for Implied Share Price

Main Body

```
Option Explicit
Option Base 1
Private PUB_SOLVE_SHARE_ARR As Variant
Function SOLVE_SHARE(ByRef TRANCHE_VAL As Variant, _
ByRef SHARE_INFO_RNG As Variant, _
ByRef ISSUE_VAL As Variant, _
ByRef CONVERTIBLE_INFO_RNG As Variant, _
ByRef EQUITY_RNG As Variant, _
ByRef BASIC_SHARE_RNG As Variant, _
Optional ByRef SHARE_PRICE_VAL As Variant, _
Optional ByRef OUTPUT As Integer = 0, _
Optional ByRef LOW_VAL As Double = 18, _
Optional ByRef UP_VAL As Double = 20) As Double

'placeholder for implied share price
Dim X_VAL As Double
'counter
Dim i As Integer

'declare array variables and transfer data from the range
Dim SHARE_INFO As Variant
Dim CONVERTIBLE_INFO As Variant
Dim EQUITY_VAL As Variant
Dim BASIC_SHARE_VAL As Variant

SHARE_INFO = SHARE_INFO_RNG
CONVERTIBLE_INFO = CONVERTIBLE_INFO_RNG
EQUITY_VAL = EQUITY_RNG
BASIC_SHARE_VAL = BASIC_SHARE_RNG

'check if the function is called by user or Muller
If IsMissing(SHARE_PRICE_VAL) Then
   GoSub LOAD_LINE:
   GoSub SOLVE_LINE:
   Exit Function
End If

'placeholders for calculations
Dim ITM_OPTION_SHARE As Double
Dim REPO_SHARE As Double
Dim OPTION_SHARE As Double
Dim CONVERTIBLE_SHARE As Double
```

Main Body Cont.

```
'calculating new share from options
ITM_OPTION_SHARE = 0
REPO_SHARE = 0
For i = 1 To TRANCHE_VAL
   If SHARE_INFO(i, 2) < SHARE_PRICE_VAL Then
      ITM_OPTION_SHARE = ITM_OPTION_SHARE + SHARE_INFO(i, 1)
      REPO_SHARE = REPO_SHARE + SHARE_INFO(i, 1) * SHARE_INFO(i, 2)
   End If
Next i
REPO_SHARE = REPO_SHARE * (-1) / SHARE_PRICE_VAL
OPTION_SHARE = ITM_OPTION_SHARE + REPO_SHARE

CONVERTIBLE_SHARE = 0
'calculating new share from convertible securities
For i = 1 To ISSUE_VAL
   If CONVERTIBLE_INFO(i, 2) < SHARE_PRICE_VAL And CONVERTIBLE_INFO(i, 2) <> 0 Then
      CONVERTIBLE_SHARE = CONVERTIBLE_SHARE + CONVERTIBLE_INFO(i, 1) / CONVERTIBLE_INFO(i,
2)
   End If
Next i

'calculate diluted share price
DILUTED_SHARE = OPTION_SHARE + CONVERTIBLE_SHARE + BASIC_SHARE_VAL
X_VAL = EQUITY_VAL / DILUTED_SHARE

'return the absolute difference
SOLVE_SHARE = Abs(X_VAL - SHARE_PRICE_VAL)
Exit Function

'populate the public array
LOAD_LINE:
   ReDim PUB_SOLVE_SHARE_ARR(1 To 6, 1 To 1)
   PUB_SOLVE_SHARE_ARR(1, 1) = TRANCHE_VAL
   PUB_SOLVE_SHARE_ARR(2, 1) = SHARE_INFO
   PUB_SOLVE_SHARE_ARR(3, 1) = ISSUE_VAL
   PUB_SOLVE_SHARE_ARR(4, 1) = CONVERTIBLE_INFO
   PUB_SOLVE_SHARE_ARR(5, 1) = EQUITY_VAL
   PUB_SOLVE_SHARE_ARR(6, 1) = BASIC_SHARE_VAL
Return
```

Main Body Cont./Objective Function

```
'solve for convergence
SOLVE_LINE:
   X_VAL = MULLER_ZERO_FUNC(LOW_VAL, UP_VAL, "SOLVE_SHARE_OBJ")
   Select Case OUTPUT
   Case 0
      SOLVE_SHARE = X_VAL
   Case 1
      SOLVE_SHARE = PUB_SOLVE_SHARE_ARR(5, 1) / X_VAL
   End Select
Return

End Function

Function SOLVE_SHARE_OBJ(ByVal X_VAL As Double) As Double

Dim Y_VAL As Double

Y_VAL = SOLVE_SHARE(PUB_SOLVE_SHARE_ARR(1, 1), _
PUB_SOLVE_SHARE_ARR(2, 1), _
PUB_SOLVE_SHARE_ARR(3, 1), _
PUB_SOLVE_SHARE_ARR(4, 1), _
PUB_SOLVE_SHARE_ARR(5, 1), _
PUB_SOLVE_SHARE_ARR(6, 1), _
X_VAL)

SOLVE_SHARE_OBJ = Y_VAL

End Function
```

Solving for Cash Balance

Main Body – Parameter Specification

```
'3 arrays, first for assets, second for liabilities, third for income statements
'--------------------------------------------------------------------------
'Asset
'1 Total Accounts Receivable
'2 Inventories
'3 Other Current Assets
'4 Total Current Assets
'5 Property, Plant & Equipment - Gross
'6 Accumulated Depreciation
'7 Net Property, Plant & Equipment
'8 Intangible Assets (Including Goodwill)
'9 Other Non-Current Assets
'10 Total Assets
'--------------------------------------------------------------------------
'Equity and Liability
'1 Accounts Payable
'2 ST Debt & Current Portion of LT Debt
'3 Other Current Liabilities
'4 Total Current Liabilities
'5 Long Term Debt
'6 Deferred Taxes
'7 Capital Lease Obligations
'8 Minority Interest Liability
'9 Other Non- Current Liabilities
'10 Total Liabilities
'11 Par + APIC - Treasury (& Other Adjustments)
'12 Retained Earnings
'13 Preferred Stock
'14 Total Equity
'--------------------------------------------------------------------------
'Net Income
'1 EBIT
'2 Interest Expense
'3 Pre-Tax Income
'4 Income taxes
'5 After Tax Adjustments
'6 Net Income
'7 EBITDA
'8 Dividends Paid
'9 Addition To RE
```

Main Body

```
Option Explicit
Option Base 1

Private PUB_SOLVE_CASH_ARR As Variant
Function SOLVE_CASH(ByRef ASSET_RNG As Variant, _
ByRef EQUITY_LIAB_RNG As Variant, _
ByRef NI_RNG As Variant, _
ByRef PREV_ASSET_VAL As Variant, _
ByRef LTD_RATIO_VAL As Variant, _
ByRef TAX_RATE_VAL As Variant, _
ByRef PREV_RE_VAL As Variant, _
ByRef PREV_CASH_VAL As Variant, _
ByRef PREV_LTD_VAL As Variant, _
ByRef EFFECTIVE_INT_RATE_VAL As Variant, _
ByRef MONEY_MARKET_INT_RATE_VAL As Variant, _
Optional ByRef CASH_VAL As Variant, _
Optional ByRef OUTPUT As Integer = 0, _
Optional ByVal LOW_VAL As Double = 9000, _
Optional ByVal UP_VAL As Double = 11000) As Double

Dim X_VAL As Double 'placeholder for cash value

If IsMissing(CASH_VAL) Then
   GoSub LOAD_LINE
   GoSub SOLVE_LINE
   Exit Function
End If

Dim SPEC_ASSET_VAL As Double
Dim SPEC_LIAB_VAL As Double
Dim NET_FIN_VAL As Double
Dim STD_VAL As Double
Dim TOT_EQUITY_VAL As Double
Dim RE_VAL As Double
Dim LTD_VAL As Double

SPEC_ASSET_VAL = PUB_SOLVE_CASH_ARR(1, 1)(1, 1) + _
            PUB_SOLVE_CASH_ARR(1, 1)(2, 1) + _
            PUB_SOLVE_CASH_ARR(1, 1)(3, 1) + _
            PUB_SOLVE_CASH_ARR(1, 1)(7, 1) + _
            PUB_SOLVE_CASH_ARR(1, 1)(8, 1) + _
            PUB_SOLVE_CASH_ARR(1, 1)(9, 1)
```

Main Body Cont.

```
LTD_VAL = (SPEC_ASSET_VAL + CASH_VAL + PUB_SOLVE_CASH_ARR(4, 1)) / 2 *
PUB_SOLVE_CASH_ARR(5, 1)
RE_VAL = PREV_RE_VAL + (PUB_SOLVE_CASH_ARR(3, 1)(1, 1) - ((LTD_VAL + PUB_SOLVE_CASH_ARR(9,
1)) / 2 * PUB_SOLVE_CASH_ARR(10, 1)) + PUB_SOLVE_CASH_ARR(8, 1) * PUB_SOLVE_CASH_ARR(11,
1)) * (1 - PUB_SOLVE_CASH_ARR(6, 1)) - PUB_SOLVE_CASH_ARR(3, 1)(8, 1)
TOT_EQUITY_VAL = PUB_SOLVE_CASH_ARR(2, 1)(11, 1) + PUB_SOLVE_CASH_ARR(2, 1)(13, 1) +
RE_VAL
SPEC_LIAB_VAL = LTD_VAL + PUB_SOLVE_CASH_ARR(2, 1)(1, 1) + PUB_SOLVE_CASH_ARR(2, 1)(3, 1) +
PUB_SOLVE_CASH_ARR(2, 1)(6, 1) + PUB_SOLVE_CASH_ARR(2, 1)(7, 1) + PUB_SOLVE_CASH_ARR(2,
1)(8, 1) + PUB_SOLVE_CASH_ARR(2, 1)(9, 1) + TOT_EQUITY_VAL
NET_FIN_VAL = (SPEC_ASSET_VAL - SPEC_LIAB_VAL)

X_VAL = Application.Max(0, -1 * NET_FIN_VAL)

SOLVE_CASH = Abs(X_VAL - CASH_VAL)
Exit Function

LOAD_LINE:
   Dim ASSET_VAL As Variant
   Dim EQUITY_LIAB_VAL As Variant
   Dim NI_VAL As Variant
   ASSET_VAL = ASSET_RNG
   EQUITY_LIAB_VAL = EQUITY_LIAB_RNG
   NI_VAL = NI_RNG
   ReDim PUB_SOLVE_CASH_ARR(1 To 11, 1 To 1)
   PUB_SOLVE_CASH_ARR(1, 1) = ASSET_VAL
   PUB_SOLVE_CASH_ARR(2, 1) = EQUITY_LIAB_VAL
   PUB_SOLVE_CASH_ARR(3, 1) = NI_VAL
   PUB_SOLVE_CASH_ARR(4, 1) = PREV_ASSET_VAL
   PUB_SOLVE_CASH_ARR(5, 1) = LTD_RATIO_VAL
   PUB_SOLVE_CASH_ARR(6, 1) = TAX_RATE_VAL
   PUB_SOLVE_CASH_ARR(7, 1) = PREV_RE_VAL
   PUB_SOLVE_CASH_ARR(8, 1) = PREV_CASH_VAL
   PUB_SOLVE_CASH_ARR(9, 1) = PREV_LTD_VAL
   PUB_SOLVE_CASH_ARR(10, 1) = EFFECTIVE_INT_RATE_VAL
   PUB_SOLVE_CASH_ARR(11, 1) = MONEY_MARKET_INT_RATE_VAL
Return

SOLVE_LINE:
   X_VAL = MULLER_ZERO_FUNC(LOW_VAL, UP_VAL, "SOLVE_CASH_OBJ")
   SOLVE_CASH = X_VAL
Return

End Function
```

Objective Function

```
Function SOLVE_CASH_OBJ(ByVal X_VAL As Double) As Double

Dim Y_VAL As Double

Y_VAL = SOLVE_CASH(PUB_SOLVE_CASH_ARR(1, 1), _
PUB_SOLVE_CASH_ARR(2, 1), _
PUB_SOLVE_CASH_ARR(3, 1), _
PUB_SOLVE_CASH_ARR(4, 1), _
PUB_SOLVE_CASH_ARR(5, 1), _
PUB_SOLVE_CASH_ARR(6, 1), _
PUB_SOLVE_CASH_ARR(7, 1), _
PUB_SOLVE_CASH_ARR(8, 1), _
PUB_SOLVE_CASH_ARR(9, 1), _
PUB_SOLVE_CASH_ARR(10, 1), _
PUB_SOLVE_CASH_ARR(11, 1), _
X_VAL)

SOLVE_CASH_OBJ = Y_VAL

End Function
```

Solving for Net Income

Main Body – Parameter Specification

```
'-------------------------------------------------------------------------
'   Non-Debt Schedule Elements
'1   Interest Rate on Excessive Cash
'2   Previous Cash Balance
'3   EBIT
'4   Tax Rate
'5   Change in Working Capital
'6   Total Depreciation
'7   Cash from Investing Activities
'8   Minium Cash Balance
'9   Other Debt
'10  Dividends
'11  Equity Issuance / (Repurchase)
'12  Commitment Fee on Unused Revolver
'13  Administrative Agent Fee
'14  Average Interest
'15  Cash Sweep
'-------------------------------------------------------------------------
'   Revolver Elements
'1   Previous Revolver Balance
'2   Revolver Interest Rate
'-------------------------------------------------------------------------
'   Term Loan Elements
'1   Term A Beginning Blanace
'2   Term A Interest Rate
'3   Term A Mandatory Payment
'4   Term B Beginning Balance
'5   Term B Interest Rate
'6   Term B Mandatory Payment
'7   Term C Beginning Balance
'8   Term C Interest Rate
'9   Term C Mandatory Payment
'-------------------------------------------------------------------------
'   Term Loan Elements
'1   ABL Facility Beginning Balance
'2   ABL Facility Coupon Rate
'3   1st Lien Beginning Balance
'4   1st Lien Coupon Rate
'5   2nd Lien Beginning Balance
'6   2nd Lien Coupon Rate
'7   Microsoft Beginning Balance
'8   Microsoft Coupon Rate
```

Main Body

```
Function SOLVE_NI(ByRef NON_DEBT_RNG As Variant, _
ByRef DEBT_REVOLVER_RNG As Variant, _
ByRef DEBT_TERM_RNG As Variant, _
ByRef DEBT_NON_TERM_RNG As Variant, _
Optional ByRef NI_VAL As Variant, _
Optional ByRef OUTPUT As Integer = 0, _
Optional LOW_VAL As Double = 2000, _
Optional UP_VAL As Double = 2300) As Double

If IsMissing(NI_VAL) Then
   GoSub LOAD_LINE:
   GoSub SOLVE_LINE:
   'clear the memory of the global array before exit the function to get ready for next time
   PUB_SOLVE_NI_ARR = Empty
   Exit Function
End If

'Declare all the variables that depends on ending cash value
Dim TOT_INT_EXP_VAL As Double
Dim INT_INCOME_VAL As Double
Dim EBT_VAL As Double

Dim OP_CASH_VAL As Double
Dim CASH_FOR_PAYMENT_VAL As Double
Dim CASH_FROM_BS_VAL As Double
Dim CASH_FOR_OP_PAYMENT_VAL As Double
Dim FIN_CASH_VAL As Double
Dim CASH_VAL As Double

Dim REVOLVER_CHANGE_VAL As Double
Dim REVOLVER_CURRENT_VAL As Double
Dim REVOLVER_INT_VAL As Double
Dim CASH_INT_VAL As Double

Dim OP_PAYMENT_VAL() As Double
Dim LEFT_OVER_CASH_VAL As Double
Dim TERM_INT_VAL() As Double
Dim NON_TERM_INT_VAL() As Double

Dim X_VAL As Double 'New Net Income Figure
Dim XX_VAL() As Double 'New Net Income Figure used for Nelder-Mead Method
ReDim XX_VAL(1 To 3, 1 To 1)
```

Main Body Cont.

```
'find cash available for optional payment = cash for debt payment + cash from balance sheet + total mandatory repayment(DEBT_TERM,3,6,9)
'find the total mandatory payment
Dim TOT_MAN_PAYMENT_VAL As Double
Dim j, k As Integer 'j is the number of term loans, k is the counter for looping
j = UBound(PUB_SOLVE_NI_ARR(3, 1)) / 3
'declare variable for optional payment
ReDim OP_PAYMENT_VAL(1 To j, 1 To 1) As Double

'calculate the total mandatory payment
TOT_MAN_PAYMENT_VAL = 0
For k = 1 To j
   TOT_MAN_PAYMENT_VAL = TOT_MAN_PAYMENT_VAL + PUB_SOLVE_NI_ARR(3, 1)((k - 1) * 3 + 3, 1)
Next k
'CASH_FOR_OP_PAYMENT_VAL = NI_VAL + PUB_SOLVE_NI_ARR(1, 1)(5, 1) + PUB_SOLVE_NI_ARR(1, 1)(6, 1) + PUB_SOLVE_NI_ARR(1, 1)(7, 1) + PUB_SOLVE_NI_ARR(1, 1)(2, 1) - PUB_SOLVE_NI_ARR(1, 1)(8, 1) + TOT_MAN_PAYMENT_VAL

'find out the drawdown/repayment from the revolver
Select Case True
   'left over cash for optional payments, use previous revolver balance (DEBT_REVOLVER,1)
   Case NI_VAL + PUB_SOLVE_NI_ARR(1, 1)(5, 1) + PUB_SOLVE_NI_ARR(1, 1)(6, 1) + PUB_SOLVE_NI_ARR(1, 1)(7, 1) + PUB_SOLVE_NI_ARR(1, 1)(2, 1) - PUB_SOLVE_NI_ARR(1, 1)(8, 1) + TOT_MAN_PAYMENT_VAL > 0
      REVOLVER_CHANGE_VAL = -1 * MIN_FUNC(NI_VAL + PUB_SOLVE_NI_ARR(1, 1)(5, 1) + PUB_SOLVE_NI_ARR(1, 1)(6, 1) + PUB_SOLVE_NI_ARR(1, 1)(7, 1) + PUB_SOLVE_NI_ARR(1, 1)(2, 1) - PUB_SOLVE_NI_ARR(1, 1)(8, 1) + TOT_MAN_PAYMENT_VAL, PUB_SOLVE_NI_ARR(2, 1)(1, 1))
   'need to borrow more from the revolver
   Case Else
      REVOLVER_CHANGE_VAL = -1 * MIN_FUNC(NI_VAL + PUB_SOLVE_NI_ARR(1, 1)(5, 1) + PUB_SOLVE_NI_ARR(1, 1)(6, 1) + PUB_SOLVE_NI_ARR(1, 1)(7, 1) + PUB_SOLVE_NI_ARR(1, 1)(2, 1) - PUB_SOLVE_NI_ARR(1, 1)(8, 1) + TOT_MAN_PAYMENT_VAL, 0)
End Select
```

Main Body Cont.

```
'find out the ending revolver balance
'REVOLVER_CURRENT_VAL = PUB_SOLVE_NI_ARR(2, 1)(1, 1) + REVOLVER_CHANGE_VAL

'find out optional payments from the 3 term loans 'optional payment for the first term loan
'optional payment method (NON_DEBT,15)
'recursive programming
LEFT_OVER_CASH_VAL = NI_VAL + PUB_SOLVE_NI_ARR(1, 1)(5, 1) + PUB_SOLVE_NI_ARR(1, 1)(6, 1) +
PUB_SOLVE_NI_ARR(1, 1)(7, 1) + PUB_SOLVE_NI_ARR(1, 1)(2, 1) - PUB_SOLVE_NI_ARR(1, 1)(8, 1) +
TOT_MAN_PAYMENT_VAL + REVOLVER_CHANGE_VAL
Select Case True
   Case PUB_SOLVE_NI_ARR(1, 1)(15, 1) = 1
      OP_PAYMENT_VAL = OP_PAYMENT_RECUR_FUNC(j, OP_PAYMENT_VAL, 0,
PUB_SOLVE_NI_ARR(3, 1), LEFT_OVER_CASH_VAL)
   Case Else
      For k = 1 To j
         OP_PAYMENT_VAL(k, 1) = 0
      Next k
End Select

'find out cash flow from financing activities
FIN_CASH_VAL = REVOLVER_CHANGE_VAL
For k = 1 To j
   FIN_CASH_VAL = FIN_CASH_VAL + OP_PAYMENT_VAL(k, 1) + PUB_SOLVE_NI_ARR(3, 1)((k - 1) * 3 +
3, 1)
Next k

'find out the new cash balance = cash from cash flow statement + previous cash balance
(NON_DEBT,2)
'CASH_VAL = FIN_CASH_VAL + NI_VAL + PUB_SOLVE_NI_ARR(1, 1)(5, 1) + PUB_SOLVE_NI_ARR(1, 1)(6,
1) + PUB_SOLVE_NI_ARR(1, 1)(7, 1) + PUB_SOLVE_NI_ARR(1, 1)(2, 1)

'find out interest expense from revolver (All the rest), interest calculation method (NON_DEBT,14)
ReDim TERM_INT_VAL(1 To j, 1 To 1) As Double
'find the number of non-term loans
Dim l As Integer
l = UBound(PUB_SOLVE_NI_ARR(4, 1)) / 2
ReDim NON_TERM_INT_VAL(1 To l, 1 To 1) As Double
```

Main Body Cont.

```
'calculate the interest payments
Select Case True
   'calculate average interest
   Case PUB_SOLVE_NI_ARR(1, 1)(14, 1) = 1
      'revolver, interest rate (DEBT_REVOLVER,2)
      'REVOLVER_INT_VAL = (PUB_SOLVE_NI_ARR(2, 1)(1, 1) + REVOLVER_CHANGE_VAL +
PUB_SOLVE_NI_ARR(2, 1)(1, 1)) / 2 * PUB_SOLVE_NI_ARR(2, 1)(2, 1)
      'committment fee on unsed revolver
      TOT_INT_EXP_VAL = (PUB_SOLVE_NI_ARR(2, 1)(1, 1) + REVOLVER_CHANGE_VAL +
PUB_SOLVE_NI_ARR(2, 1)(1, 1)) / 2 * PUB_SOLVE_NI_ARR(2, 1)(2, 1) + PUB_SOLVE_NI_ARR(1, 1)(12, 1)
* (PUB_SOLVE_NI_ARR(2, 1)(3, 1) - (PUB_SOLVE_NI_ARR(2, 1)(1, 1) * 2 + REVOLVER_CHANGE_VAL) / 2)
      'term loan
      'Interest *(Beginning Balance + Mandatory Payment + Optional Payment + Beginning Balance)
      For k = 1 To j
         TERM_INT_VAL(k, 1) = PUB_SOLVE_NI_ARR(3, 1)(3 * (k - 1) + 2, 1) * (2 * PUB_SOLVE_NI_ARR(3,
1)(3 * (k - 1) + 1, 1) + PUB_SOLVE_NI_ARR(3, 1)((k - 1) * 3 + 3, 1) + OP_PAYMENT_VAL(k, 1)) / 2
         TOT_INT_EXP_VAL = TOT_INT_EXP_VAL + TERM_INT_VAL(k, 1)
      Next k
   'calculate ending balance interest
   Case Else
      'revolver
      'REVOLVER_INT_VAL = (PUB_SOLVE_NI_ARR(2, 1)(1, 1) + REVOLVER_CHANGE_VAL) *
PUB_SOLVE_NI_ARR(2, 1)(2, 1)
      TOT_INT_EXP_VAL = (PUB_SOLVE_NI_ARR(2, 1)(1, 1) + REVOLVER_CHANGE_VAL) *
PUB_SOLVE_NI_ARR(2, 1)(2, 1)
      'term loan only ending balance
      For k = 1 To j
         'TERM_INT_VAL(k, 1) = PUB_SOLVE_NI_ARR(3, 1)(3 * (k - 1) + 2, 1) * (PUB_SOLVE_NI_ARR(3,
1)(3 * (k - 1) + 1, 1) + PUB_SOLVE_NI_ARR(3, 1)((k - 1) * 3 + 3, 1) + OP_PAYMENT_VAL(k, 1))
         TOT_INT_EXP_VAL = TOT_INT_EXP_VAL + PUB_SOLVE_NI_ARR(3, 1)(3 * (k - 1) + 2, 1) *
(PUB_SOLVE_NI_ARR(3, 1)(3 * (k - 1) + 1, 1) + PUB_SOLVE_NI_ARR(3, 1)((k - 1) * 3 + 3, 1) +
OP_PAYMENT_VAL(k, 1))
      Next k
End Select

'non-term loan interest, since they are perpetual in this model
'Interest rate * Beginning Balance
For k = 1 To l
   'NON_TERM_INT_VAL(k, 1) = PUB_SOLVE_NI_ARR(4, 1)(2 * (k - 1) + 2, 1) * PUB_SOLVE_NI_ARR(4,
1)(2 * (k - 1) + 1, 1)
   TOT_INT_EXP_VAL = TOT_INT_EXP_VAL + PUB_SOLVE_NI_ARR(4, 1)(2 * (k - 1) + 2, 1) *
PUB_SOLVE_NI_ARR(4, 1)(2 * (k - 1) + 1, 1)
Next k
```

Main Body Cont.

```
'add agent fee and amortization of financing fee
TOT_INT_EXP_VAL = TOT_INT_EXP_VAL + PUB_SOLVE_NI_ARR(1, 1)(13, 1) + PUB_SOLVE_NI_ARR(1, 1)(16, 1)
'find interest income
'INT_INCOME_VAL = (FIN_CASH_VAL + NI_VAL + PUB_SOLVE_NI_ARR(1, 1)(5, 1) + PUB_SOLVE_NI_ARR(1, 1)(6, 1) + PUB_SOLVE_NI_ARR(1, 1)(7, 1) + PUB_SOLVE_NI_ARR(1, 1)(2, 1) + PUB_SOLVE_NI_ARR(1, 1)(2, 1)) / 2 * PUB_SOLVE_NI_ARR(1, 1)(1, 1)

'find EBT
EBT_VAL = PUB_SOLVE_NI_ARR(1, 1)(3, 1) - TOT_INT_EXP_VAL - ((FIN_CASH_VAL + NI_VAL + PUB_SOLVE_NI_ARR(1, 1)(5, 1) + PUB_SOLVE_NI_ARR(1, 1)(6, 1) + PUB_SOLVE_NI_ARR(1, 1)(7, 1) + PUB_SOLVE_NI_ARR(1, 1)(2, 1) + PUB_SOLVE_NI_ARR(1, 1)(2, 1)) / 2 * PUB_SOLVE_NI_ARR(1, 1)(1, 1))

'find NI
X_VAL = EBT_VAL * (1 - PUB_SOLVE_NI_ARR(1, 1)(4, 1))
SOLVE_NI = Abs(X_VAL - NI_VAL)
Exit Function

'Load the constants that doesn't change during each iteration
LOAD_LINE:
   Dim NON_DEBT_VAL As Variant
   Dim DEBT_REVOLVER_VAL As Variant
   Dim DEBT_TERM_VAL As Variant
   Dim DEBT_NON_TERM_VAL As Variant
   NON_DEBT_VAL = NON_DEBT_RNG 'array of non-debt related constants required in the circular reference calculation
   DEBT_REVOLVER_VAL = DEBT_REVOLVER_RNG 'array of revolver constants required in the circular reference calculation
   DEBT_TERM_VAL = DEBT_TERM_RNG
   DEBT_NON_TERM_VAL = DEBT_NON_TERM_RNG

   ReDim PUB_SOLVE_NI_ARR(1 To 4, 1 To 1)
   PUB_SOLVE_NI_ARR(1, 1) = NON_DEBT_VAL
   PUB_SOLVE_NI_ARR(2, 1) = DEBT_REVOLVER_VAL
   PUB_SOLVE_NI_ARR(3, 1) = DEBT_TERM_VAL
   PUB_SOLVE_NI_ARR(4, 1) = DEBT_NON_TERM_VAL
Return
```

Main Body Cont./ Optional Payment Calculation

```
'Iteration process using Nelder-Mead's method
SOLVE_LINE:
   XX_VAL = NELDER_MEAD(LOW_VAL, UP_VAL, "SOLVE_NI_OBJ") 'XX_VAL(1,1) is the stable point for
Net Income
   Select Case True
      Case OUTPUT = 0
         SOLVE_NI = XX_VAL(1, 1)
      Case Else
      Debug.Print "!"
      Exit Function
   End Select
Return
End Function

'Recursive Formula for Calculating Optional Payments
Function OP_PAYMENT_RECUR_FUNC(ByVal NUM_PAYMENT_VAL As Integer, ByRef
OP_PAYMENT_VAL As Variant, ByRef COUNTER_VAL As Integer, ByRef DEBT_VAL As Variant, ByRef
CASH_FOR_OP_PAYMENT_VAL As Double) As Variant
Dim i As Integer 'counter variable for looping
Dim CURRENT_OP_PAYMENT_VAL As Double 'holder for current optional payments

Select Case True
   Case COUNTER_VAL < NUM_PAYMENT_VAL
      Select Case True
         Case COUNTER_VAL = 0
            OP_PAYMENT_VAL(COUNTER_VAL + 1, 1) = -MIN_FUNC(DEBT_VAL(1 + COUNTER_VAL * 3, 1)
+ DEBT_VAL(3 + COUNTER_VAL * 3, 1), CASH_FOR_OP_PAYMENT_VAL)
         Case Else
            CURRENT_OP_PAYMENT_VAL = 0
            For i = 1 To COUNTER_VAL
               CURRENT_OP_PAYMENT_VAL = CURRENT_OP_PAYMENT_VAL + OP_PAYMENT_VAL(i, 1)
            Next i
            OP_PAYMENT_VAL(COUNTER_VAL + 1, 1) = -MIN_FUNC(DEBT_VAL(1 + COUNTER_VAL * 3, 1)
+ DEBT_VAL(3 + COUNTER_VAL * 3, 1), CASH_FOR_OP_PAYMENT_VAL +
CURRENT_OP_PAYMENT_VAL)
         End Select
      COUNTER_VAL = COUNTER_VAL + 1
      OP_PAYMENT_RECUR_FUNC = OP_PAYMENT_RECUR_FUNC(NUM_PAYMENT_VAL,
OP_PAYMENT_VAL, COUNTER_VAL, DEBT_VAL, CASH_FOR_OP_PAYMENT_VAL)
   Case COUNTER_VAL = NUM_PAYMENT_VAL
      OP_PAYMENT_RECUR_FUNC = OP_PAYMENT_VAL
      Exit Function
End Select

End Function
```

Objective Function /Search for Minimum Function

```
Function SOLVE_NI_OBJ(ByVal X_VAL As Double) As Double

Dim Y_VAL As Double

Y_VAL = SOLVE_NI(PUB_SOLVE_NI_ARR(1, 1), PUB_SOLVE_NI_ARR(2, 1), PUB_SOLVE_NI_ARR(3, 1),
PUB_SOLVE_NI_ARR(4, 1), X_VAL)

SOLVE_NI_OBJ = Y_VAL

End Function

Function MIN_FUNC(ByVal NUM_A As Double, ByVal NUM_B As Double, ParamArray REST_NUM() As
Variant) As Double

Dim TOT_NUM_VAL As Double

TOT_NUM_VAL = UBound(REST_NUM) + 2

Select Case True
   Case NUM_A < NUM_B
      MIN_FUNC = NUM_A
   Case NUM_A > NUM_B
      MIN_FUNC = NUM_B
End Select

Select Case True
   Case TOT_NUM_VAL > 2
      Dim i As Integer
      For i = 1 To (TOT_NUM_VAL - 2)
         Select Case True
            Case MIN_FUNC > REST_NUM(i)
               MIN_FUNC = REST_NUM(i)
         End Select
      Next
      Exit Function
   Case Else
      Exit Function
End Select

End Function
```

Nelder-Mead Process

```
'nelder mead procedure
Function NELDER_MEAD(ByRef LOW_VAL, ByRef HIGH_VAL, ByRef FUNC_NAME_STR As String,
Optional ByRef nLOOP = 600, Optional ByRef EPSILON = 10 ^ -4)

'declare initial simplex as 0 and -1
Dim INITIAL_SIMPLEX() As Double
ReDim INITIAL_SIMPLEX(1 To 1, 1 To 2)
INITIAL_SIMPLEX(1, 1) = LOW_VAL
INITIAL_SIMPLEX(1, 2) = HIGH_VAL

'declare counter variable
Dim i As Integer

'declare result array
Dim RESULT() As Double
ReDim RESULT(1 To 1, 1 To 3)

'declare the decision variable on distinguishing Reflection/Expansion and Cotraction/Shrink
Dim SIMPLEX_DEC As Boolean
'declare variables required for the nelder-mead method
Dim REFLECTION, EXPANSION, CONTRACTION As Double
Dim GOOD_POINT, BAD_POINT, TEMP_POINT As Double

'decide which of the initial guess is good point
If Abs(Excel.Application.Run(FUNC_NAME_STR, INITIAL_SIMPLEX(1, 1))) >
Abs(Excel.Application.Run(FUNC_NAME_STR, INITIAL_SIMPLEX(1, 2))) Then
   GOOD_POINT = INITIAL_SIMPLEX(1, 2)
   BAD_POINT = INITIAL_SIMPLEX(1, 1)
Else
   GOOD_POINT = INITIAL_SIMPLEX(1, 1)
   BAD_POINT = INITIAL_SIMPLEX(1, 2)
End If
```

Nelder-Mead Process Cont.

```
'start looping for the simplex method
For i = 1 To nLOOP
   'recaliber good point and bad point
   If Abs(Excel.Application.Run(FUNC_NAME_STR, BAD_POINT)) <
Abs(Excel.Application.Run(FUNC_NAME_STR, GOOD_POINT)) Then
      TEMP_POINT = BAD_POINT
      BAD_POINT = GOOD_POINT
      GOOD_POINT = TEMP_POINT
   End If
   'calculate the reflection point
   REFLECTION = GOOD_POINT + (GOOD_POINT - BAD_POINT)
   'check if the reflection point is better than the bad point; if so, check whether to reflect or to
expand
   If Abs(Excel.Application.Run(FUNC_NAME_STR, REFLECTION)) <
Abs(Excel.Application.Run(FUNC_NAME_STR, BAD_POINT)) Then
      'if good point is better than reflection point, reflect the simplex
      If Abs(Excel.Application.Run(FUNC_NAME_STR, GOOD_POINT)) <
Abs(Excel.Application.Run(FUNC_NAME_STR, REFLECTION)) Then
         BAD_POINT = REFLECTION
      Else
         'compute expansion point
         EXPANSION = REFLECTION + (REFLECTION - GOOD_POINT)
         'if the expansion point is better than the good point, change bad point to expansion point
         If Abs(Excel.Application.Run(FUNC_NAME_STR, EXPANSION)) <
Abs(Excel.Application.Run(FUNC_NAME_STR, GOOD_POINT)) Then
            BAD_POINT = EXPANSION
         Else
            BAD_POINT = REFLECTION
         End If
      End If
   'if the reflection point is worse than the good point, shrink/contract
   Else
      CONTRACTION = GOOD_POINT + 0.5 * (BAD_POINT - GOOD_POINT)
      BAD_POINT = CONTRACTION
   End If
   'exit the function if converngece occurs
   If Abs(Excel.Application.Run(FUNC_NAME_STR, GOOD_POINT)) < EPSILON Then
      RESULT(1, 1) = GOOD_POINT
      RESULT(1, 2) = Excel.Application.Run(FUNC_NAME_STR, GOOD_POINT)
      NELDER_MEAD = RESULT
      Exit Function
   End If
Next i
```

Nelder-Mead Process Cont.

```
'check if there is convergence
If i = nLOOP + 1 And (Abs(Excel.Application.Run(FUNC_NAME_STR, GOOD_POINT))) > EPSILON Then
   RESULT(1, 3) = -1 'no convergence
End If
'populate results
RESULT(1, 1) = GOOD_POINT
RESULT(1, 2) = Excel.Application.Run(FUNC_NAME_STR, GOOD_POINT)
NELDER_MEAD = RESULT

End Function
```

Muller's Method – Courtesy of Rafael Nicolas Fermin Cota

```
'*************************************************************************************
'FUNCTION     : MULLER_ZERO_FUNC
'DESCRIPTION  : Calculates the values necessary to achieve a specific goal -
'implements the Muller's method
'LIBRARY      : OPTIMIZATION
'GROUP        : UNIVAR_ZERO
'ID           : 012
'AUTHOR       : RAFAEL NICOLAS FERMIN COTA
'LAST UPDATE  : 12/08/2008
'*************************************************************************************
Function MULLER_ZERO_FUNC(ByVal LOWER_VAL As Double, _
ByVal UPPER_VAL As Double, _
ByVal FUNC_NAME_STR As String, _
Optional ByRef CONVERG_VAL As Integer, _
Optional ByRef COUNTER As Long, _
Optional ByVal nLOOPS As Long = 600, _
Optional ByVal tolerance As Double = 1)

Dim ATEMP_VAL As Double
Dim BTEMP_VAL As Double
Dim CTEMP_VAL As Double
Dim DTEMP_VAL As Double

Dim TEMP_MID As Double
Dim TEMP_MULT As Double
Dim TEMP_GRAD As Double

Dim TEMP_FUNC As Double
Dim FIRST_FUNC As Double
Dim SECOND_FUNC As Double

On Error GoTo ERROR_LABEL
```

Muller's Method Cont. – Courtesy of Rafael Nicolas Fermin Cota

```
'-------------------------------------------------------------------------
' MULLER implements Muller's method
'
'  Parameters:
'
'    Input/output, real X, X1, X2.
'    On input, three distinct points that start the method.
'    On output, X is an approximation to a root of the equation
'    which satisfies abs ( F(X) ) < ABSERR, and X1 and X2 are the
'    previous estimates.
'-------------------------------------------------------------------------

  CONVERG_VAL = 0
  COUNTER = 0
  TEMP_MID = (LOWER_VAL + UPPER_VAL) / 2

  SECOND_FUNC = Excel.Application.Run(FUNC_NAME_STR, UPPER_VAL)
  FIRST_FUNC = Excel.Application.Run(FUNC_NAME_STR, LOWER_VAL)
  TEMP_FUNC = Excel.Application.Run(FUNC_NAME_STR, TEMP_MID)
```

Muller's Method Cont. – Courtesy of Rafael Nicolas Fermin Cota

```
' Iteration loop:
  Do
' If the error tolerance is satisfied, then exit.
    If (Abs(TEMP_FUNC) <= tolerance) Then
      MULLER_ZERO_FUNC = TEMP_MID
      Exit Function
    End If
    COUNTER = COUNTER + 1
    If (COUNTER > nLOOPS) Then
      CONVERG_VAL = 2
      MULLER_ZERO_FUNC = TEMP_MID
      Exit Function
    End If
    DTEMP_VAL = (TEMP_MID - LOWER_VAL) / (LOWER_VAL - UPPER_VAL)
    'variabile normalizzata   0 < DTEMP < 1
    ATEMP_VAL = DTEMP_VAL * TEMP_FUNC - DTEMP_VAL * (1 + DTEMP_VAL) * FIRST_FUNC + _
          DTEMP_VAL ^ 2 * SECOND_FUNC
    BTEMP_VAL = (2 * DTEMP_VAL + 1) * TEMP_FUNC - (1 + DTEMP_VAL) ^ 2 * FIRST_FUNC + _
       DTEMP_VAL ^ 2 * SECOND_FUNC
    CTEMP_VAL = (1 + DTEMP_VAL) * TEMP_FUNC

    TEMP_MULT = BTEMP_VAL ^ 2 - 4 * ATEMP_VAL * CTEMP_VAL
    If TEMP_MULT < 0 Then TEMP_MULT = 0
    TEMP_MULT = Sqr(TEMP_MULT)
    If (BTEMP_VAL < 0) Then: TEMP_MULT = -TEMP_MULT

' Set the increment.
    TEMP_GRAD = -(TEMP_MID - LOWER_VAL) * 2 * CTEMP_VAL / (BTEMP_VAL + TEMP_MULT)
' Remember current data for next step.
    UPPER_VAL = LOWER_VAL
    SECOND_FUNC = FIRST_FUNC
    LOWER_VAL = TEMP_MID
    FIRST_FUNC = TEMP_FUNC
'
' Update the iterate and function values.
'
    TEMP_MID = TEMP_MID + TEMP_GRAD
    TEMP_FUNC = Excel.Application.Run(FUNC_NAME_STR, TEMP_MID)

  Loop

Exit Function
ERROR_LABEL:
    MULLER_ZERO_FUNC = -1000000
End Function
```