CHAPTER 5

Newton-Raphson Method

5.1. Newton-Raphson Method for System of Equations

The use of EXCEL SOLVER is quite convenient in finding a solution $\{ x_1, \ldots , x_n \}$ for the zero of an objective function where $g(x_1, \ldots , x_n) = 0$. For multiple objective functions with the same variables, however, it is not applicable for SOLVER to solve simultaneously the zero of a system of equations given by

$$\begin{gathered} g_1(x_1, \ldots , x_n) = 0 \\ \ldots \\ g_n(x_1, \ldots , x_n) = 0 \end{gathered} \tag{5.1}$$

We consider here a powerful numerical procedure known as Newton-Raphson method[1] capable of handling such problem. The objective in this chapter is to build a generic VBA routine for the Newton-Raphson procedure that will definitely be useful in the implementations of the financial models to be discussed in forthcoming chapters.

Algebraically, the method derives from the familiar Taylor series expansion of a function in the neighborhood of a point. Suppose we want to determine the zero of a function with one variable where $g(x^{soln}) = 0$. Consider a trial guess of the solution x^{old} where the error involved ε^{old} is presumably small. The Taylor expansion of the function in the neighborhood of x^{old} can be written as

$$g(x^{old} + \varepsilon^{old}) = g(x^{old}) + g'(x^{old})\, \varepsilon^{old} + \ldots$$

where the higher-order terms in the series are unimportant. The entire expression must vanish by definition as $x^{soln} = x^{old} + \varepsilon^{old}$ is the zero of the function. Hence, we can estimate the error in the trial solution as $\varepsilon^{old} \cong -\, g(x^{old})/g'(x^{old})$ and accordingly update the guess to be

$$x^{new} = x^{old} - \frac{g(x^{old})}{g'(x^{old})} \tag{5.2}$$

The error involved $x^{soln} = x^{new} + \varepsilon^{new}$ after the update will be much smaller as it can be shown that the size of ε^{new} is in quadratic power[2] of ε^{old}. In equation (5.2), it requires the evaluation of both the function and the derivative at the trial guess, and generates an update that is much closer to the solution. Under a defined precision Δx, the derivative can be approximated using the numerical difference given by

$$g'(x) \cong \frac{g(x + \Delta x) - g(x)}{\Delta x}$$

The Newton-Raphson procedure should proceed iteratively using (5.2) and stop literally when the improvement has reached the precision limit of $| \, x^{new} - x^{old} \, | \leq \Delta x$.

[1] See Chapter 9 of W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, *Numerical Recipes in C : The Art of Scientific Computing*, 2nd Edition, Cambridge University Press, 1997.

[2] From (5.2), we have $\varepsilon^{new} = \varepsilon^{old} + g(x^{soln} - \varepsilon^{old})/g'(x^{soln} - \varepsilon^{old})$. We can expand the terms g and g' around x^{soln} to get $\varepsilon^{new} \approx -\frac{1}{2}\, [\, g''(x^{soln})/g'(x^{soln})(\varepsilon^{old})^2$.

The method can readily be generalized to multiple dimensions such as the system of equations in (5.1). Consider a trial guess of the zero solution $\{ x_1^{old}, \dots , x_n^{old} \}$ with errors $\{ \varepsilon_1^{old}, \dots , \varepsilon_n^{old} \}$. The Taylor expansions of the multivariable functions in the neighborhood of the trial guess can be written as

$$g_i(x_1^{old} + \varepsilon_1^{old}, \dots , x_n^{old} + \varepsilon_n^{old}) = g_i(x_1^{old}, \dots , x_n^{old}) + \sum_{j=1}^{n} [\, \partial g_i(x_1^{old}, \dots , x_n^{old})/\partial x_j \,]\varepsilon_j^{old} + \dots , \quad i = 1, \dots , n.$$

They must all vanish simultaneously as $\{ x_1^{old} + \varepsilon_1^{old}, \dots , x_n^{old} + \varepsilon_n^{old} \}$ is the zero of the system. Again, we can estimate the errors in the trial solution and update the guess to be

$$\boldsymbol{x}^{new} = \boldsymbol{x}^{old} - \boldsymbol{\Omega}^{-1}(x_1^{old}, \dots , x_n^{old})\, \boldsymbol{g}(x_1^{old}, \dots , x_n^{old}) \qquad (5.3)$$

where $\boldsymbol{x} = \{ x_1, \dots , x_n \}$ and $\boldsymbol{g} = \{ g_1, \dots , g_n \}$ are defined to be column vectors in the matrix equation and $\boldsymbol{\Omega}$ is a $n{\times}n$ square matrix given by

$$\boldsymbol{\Omega}(x_1, \dots , x_n) = \begin{bmatrix} \partial g_1(x_1, \dots , x_n)/\partial x_1 & \cdots & \partial g_1(x_1, \dots , x_n)/\partial x_n \\ : & & : \\ \partial g_n(x_1, \dots , x_n)/\partial x_1 & \cdots & \partial g_n(x_1, \dots , x_n)/\partial x_n \end{bmatrix} \qquad (5.4)$$

The partial derivatives in (5.4) can be approximated using the numerical difference under a defined precision Δx as

$$\frac{\partial}{\partial x_j} g_i(x_1, \dots , x_n) \cong \frac{g_i(x_1, \dots , x_j + \Delta x, \dots , x_n) - g_i(x_1, \dots , x_n)}{\Delta x} \qquad (5.5)$$

Similarly, the Newton-Raphson procedure should proceed iteratively using (5.3) and stop when the improvements for all variables have reached the precision limit of $| x_i^{new} - x_i^{old} | \le \Delta x$.

5.2. VBA Routine

We want to build here a generic VBA routine called NewtonRaphson( ) that implements the Newton-Raphson procedure as given by equations (5.3), (5.4), and (5.5) in an arbitrary dimension. The objective functions should be defined external to the routine such that it is applicable to different kinds of problems. The pseudo code of NewtonRaphson( ) is given by Code 5.1. The routine reads in trial values of the variables $\{ x_1, \dots , x_n \}$ together with a designated precision Δx in the solution. It returns the last updated values as the solution of the problem through the iteration procedure bounded by the precision limit. The iteration starts off from the trial values (initiate $\boldsymbol{x}^{old} = \boldsymbol{x}$ with the trial values when $Nitr = 1$) and will continue to update the variables ($\boldsymbol{x} = \boldsymbol{x}^{old} - \boldsymbol{\Omega}^{-1}\boldsymbol{g}$) when the precision limit has not yet been reached (*precflag* = FALSE). To prevent entering a dead loop, we impose an upper limit of $Nitrmax = 1000$ on the maximum number of iterations to be performed. As reference, NewtonRaphson( ) returns the status of the precision flag at exit to indicate whether it is terminated by the iteration limit where the designated precision has not been met. For cross-checking purpose, the routine also returns the maximum deviation from zero among all the functions as evaluated at the point of exit.

At each update, the iteration requires an ad hoc evaluation of both the objective functions and their partial derivatives. This can be done through an external routine called FunctionArray( ) that defines the kind of problem at hand and returns array of function values $\{ g_1, \dots , g_n \}$ evaluated at specific input values of $\{ x_1, \dots , x_n \}$. Partial derivatives in the entries of $\boldsymbol{\Omega}$ can be estimated by making two consecutive calls to FunctionArray( ) based on the numerical difference in (5.5). While the double-loop structure with *j* and *i* will run through all its entries, the interior loop with *k* will shift only the appropriate term in the array { $x_1, \ldots, x_n$ } by Δx before making a second call to FunctionArray( ) for changes in function values.

The VBA code of NewtonRaphson( ) is given by Code 5.2. For the calculation of variable update, we have again used the routine SolveAxb( ) to calculate the shift $\mathbf{\Omega}^{-1}\boldsymbol{g}$. During the iteration, the value of *precflag* under the logical conjunction of an array of conditions can be determined by looping through each of the condition. The precision limit has been reached (*precflag* = .TRUE.) if there is no violation on either one of these conditions. Consider now the use of the NewtonRaphson( ) routine in the following examples.

***Example* 5.1** :

Suppose we want to solve simultaneously the zero of a pair of functions ($n = 2$) given by

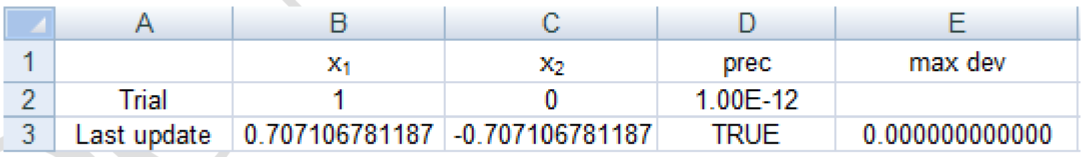
$$g_1(x_1, x_2) = x_1^2 + x_2^2 - 1$$

$$g_2(x_1, x_2) = x_1 + x_2$$

Here, the FunctionArray( ) routine should be able to read in specific values of { x_1, x_2 } and return the corresponding function values { g_1, g_2 }. The Newton-Raphson solution of this problem will have the following VBA structure.

```
Sub Test()
  Dim x(1 To 2) As Double, n As Integer, prec As Double, precFlag As Boolean, maxDev As Double
  n = 2
  x(1) = Range("B2").Value
  x(2) = Range("C2").Value
  prec = Range("D2").Value
  Call NewtonRaphson(n, prec, x, precFlag, maxDev)
  Range("B3:C3") = x
  Range("D3") = precFlag
  Range("E3") = maxDev
End Sub

Sub FunctionArray(n As Integer, x() As Double, ByRef g() As Double)
  g(1) = x(1) ^ 2 + x(2) ^ 2 - 1
  g(2) = x(1) + x(2)
End Sub
```



	A	B	C	D	E
1		x_1	x_2	prec	max dev
2	Trial	1	0	1.00E-12	
3	Last update	0.707106781187	-0.707106781187	TRUE	0.000000000000

***Example* 5.2** :

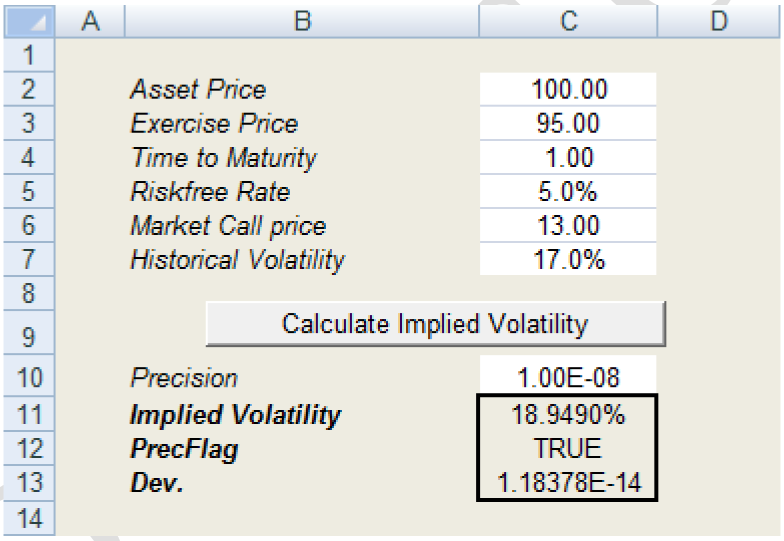
A practical example of interest in finance is the estimation of the so-called implied volatility in option pricing where the theoretical Black-Scholes value matches the market price. Taking the volatility parameter σ to be the only variable ($n = 1$), we want to solve the zero of the following function that represents the difference between, for example, the call option prices based on Black-Scholes formula and the market.

$$g(\sigma) = S\,N(d) - K\,e^{-rT}\,N(d - \sigma\sqrt{T}) - c_{market} \quad , \qquad d = \frac{\ln(S/K) + (\,r + \tfrac{1}{2}\sigma^2\,)T}{\sigma\sqrt{T}}$$

Here, *S* is the asset price, *r* is the risk-free interest rate, *K* and *T* are the strike price and maturity of the option, respectively. The term c_{market} is the market option price with the same strike and maturity. The mathematical function $N(x)$ is the cumulative normal distribution with zero mean and unit standard deviation. In VBA, we can simply call the EXCEL function **NORMSDIST** for values of $N(x)$. The VBA coding for this routine is given as follows:

```
Sub calImpVol()
   Dim sigma(1 To 1) As Double, n As Integer, prec As Double, precFlag As Boolean, maxDev As Double
   n = 1
   sigma(1) = Range("C7").Value
   prec = Range("C10").Value
   Call NewtonRaphson(n, prec, sigma, precFlag, maxDev)
   Range("C11") = sigma(1)
   Range("C12") = precFlag
   Range("C13") = maxDev
End Sub

Sub FunctionArray(n As Integer, sigma() As Double, ByRef g() As Double)
   Dim assetPrice As Double, exercisePrice As Double, timeMaturity As Double
   Dim riskFree As Double, marketCallPrice As Double, d As Double
   assetPrice = Range("C2").Value
   exercisePrice = Range("C3").Value
   timeMaturity = Range("C4").Value
   riskFree = Range("C5").Value
   marketCallPrice = Range("C6").Value
   d = Log(assetPrice / exercisePrice) + (riskFree + 0.5 * sigma(1) ^ 2) * timeMaturity
   d = d / (sigma(1) * timeMaturity ^ 0.5)
   With Application.WorksheetFunction
      g(1) = assetPrice * .NormSDist(d) - _
           exercisePrice * Exp(-riskFree * timeMaturity) * _
           .NormSDist(d - sigma(1) * timeMaturity ^ 0.5) - marketCallPrice
   End With
End Sub
```



	A	B	C	D
1				
2		*Asset Price*	100.00	
3		*Exercise Price*	95.00	
4		*Time to Maturity*	1.00	
5		*Riskfree Rate*	5.0%	
6		*Market Call price*	13.00	
7		*Historical Volatility*	17.0%	
8				
9		Calculate Implied Volatility		
10		*Precision*	1.00E-08	
11		***Implied Volatility***	18.9490%	
12		***PrecFlag***	TRUE	
13		***Dev.***	1.18378E-14	
14				

```
$NewtonRaphson( n , \Delta x , \mathbf{x}(1{:}n) , precflag , maxdev )$

# define the maximum number of iterations
    $Nitrmax = 1000$

# iterate to a new point when both the precision and iteration limits have not yet been reached
    For( $Nitr = 1$ to $Nitrmax$ ){

# initiate the array of variables for iteration
                        $\mathbf{x}^{old} ( 1 : n ) = \mathbf{x}( 1 : n )$

# determine the function values
                        call $FunctionArray( n , x^{old}(1{:}n) , g(1{:}n) )$

# determine the matrix $\mathbf{\Omega}$ by making another call to FunctionArray with shifted $x$
                        For ( $j = 1$ to $n$ ) {  For ( $k = 1$ to $n$ ) {  $x^{shift}(k) = x^{old}(k) + \Delta x\, \delta_{jk}$  }

                              call $FunctionArray( n , x^{shift}(1{:}n) , g^{shift}(1{:}n) )$

                              For ( $i = 1$ to $n$ ) {  $\Omega(i, j) = [ g^{shift}(i) - g(i) ] / \Delta x$  }
                             }

# iterate and update the column array of variables
                        $\mathbf{x}( 1 : n ) = \mathbf{x}^{old}( 1 : n ) - \mathbf{\Omega}^{-1}( 1 : n , 1 : n )\, \mathbf{g}( 1 : n )$

# check the precision limit and update the precision flag
                        $precflag = \mathrm{AND}( | x(1) - x^{old}(1) | \le \Delta x , \dots , | x(n) - x^{old}(n) | \le \Delta x )$

                        If( $precflag$ ) then exit $Nitr$

                           }

# determine at exit the maximum deviation from zero among all the functions
    call $FunctionArray( n , x(1{:}n) , g(1{:}n) )$

    $maxdev = \mathrm{MAX}( | g_1 | , \dots , | g_N | )$
```

Code 5.1: Pseudo code of the NewtonRaphson( ) routine

```
Sub NewtonRaphson(n As Integer, prec As Double, ByRef x() As Double, ByRef precFlag As Boolean, ByRef maxDev As Double)
    Const nItrMax As Integer = 1000
    Dim xOld() As Double: ReDim xOld(1 To n)
    Dim xShift() As Double: ReDim xShift(1 To n)
    Dim gShift() As Double: ReDim gShift(1 To n)
    Dim g() As Double: ReDim g(1 To n)
    Dim omega() As Double: ReDim omega(1 To n, 1 To n)
    Dim Dx() As Double: ReDim Dx(1 To n)

    Dim i As Integer, j As Integer, k As Integer, nItr As Integer
    For nItr = 1 To nItrMax
        'initiate the array of variables and determine the function values
        For i = 1 To n: xOld(i) = x(i): Next i
        Call FunctionArray(n, xOld, g)
        'determine the matrix omega
        For j = 1 To n
            For k = 1 To n: xShift(k) = xOld(k) + prec * IIf(j = k, 1, 0): Next k
            Call FunctionArray(n, xShift, gShift)
            For i = 1 To n: omega(i, j) = (gShift(i) - g(i)) / prec: Next i
        Next j

        'iterate and update the array of variables
        Call SolveAxb(omega, g, Dx, n, 1, 1, 1)
        For i = 1 To n: x(i) = xOld(i) - Dx(i): Next i

        'check the precision limit and update the precision flag
        For i = 1 To n
            If Abs(x(i) - xOld(i)) <= prec Then
             precFlag = True
            Else
             precFlag = False
             Exit For
            End If
        Next i
        If precFlag Then Exit For
    Next nItr
    'determine the maximum deviation at exit
    Call FunctionArray(n, x, g)
    maxDev = 0
    For i = 1 To n
       If Abs(g(i)) > maxDev Then maxDev = Abs(g(i))
    Next i
End Sub
```

Code 5.2: VBA code of the NewtonRaphson( ) routine