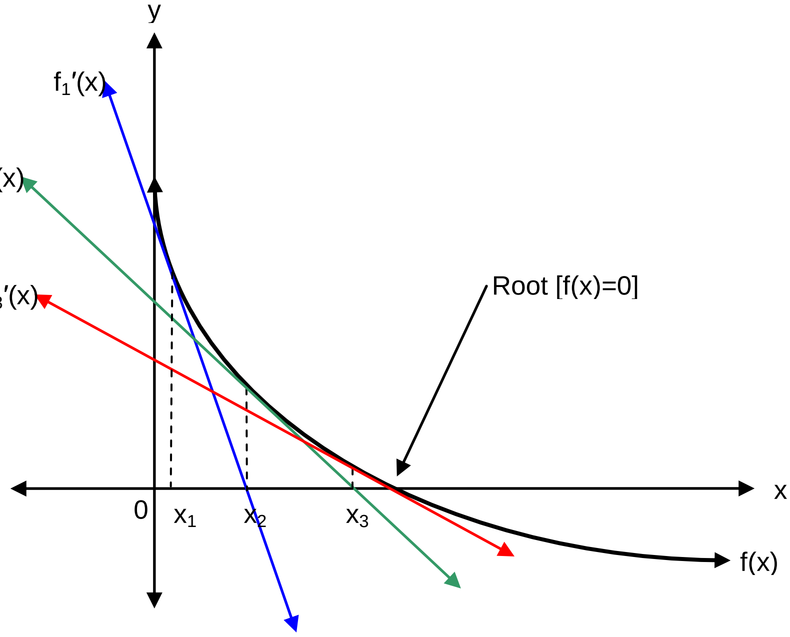
Newton-Raphson Approximation with Applications

by Hammond Mason



Consider the above function f(x). The point at which f(x) intersects the x-axis is the value of x for which the function is zero. This x value is called the root of f(x). In the situation where f(x) represents the NPV of a set of cash flows discounted at a rate equal to x, the root of f(x) is commonly known as the Internal Rate of Return (IRR). The root may be determined exactly if the equation for f(x) is known and is not of a very high order, eg. quadratic or cubic. When the equation is unknown – as is often the case in finance – the root can be *approximated* using the Newton-Raphson algorithm.

The Newton-Raphson Algorithm

Knowing the equation for f(x) we find its first derivative f'(x) at a point estimated to be at or near the root. In our example above, the 'best guess' value we chose was x_1. Knowing that $f_1'(x_1)$ is equal to the gradient of the line tangential to f(x) at x_1 we can easily find point x_2 using the equation:

$$gradient = \frac{rise}{run}$$

$$\therefore f'(x) = \frac{\Delta y}{\Delta x}$$

$$\therefore f'(x_1) = \frac{y_2 - y_1}{x_2 - x_1}$$

$$\therefore f'(x_1) = \frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

Knowing that $f(x_2) = 0$ we can rearrange the above to:

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

And solve for x_2. We then repeat the process using x_2 in order to find x_3. And so on. Using this iterative method, we very quickly come close to finding the root of f(x). We come *close* to the root but never actually reach it. Our approximation will always need to come within a degree of error as specified by the user – eg. the number of decimal places, significant figures, etc.

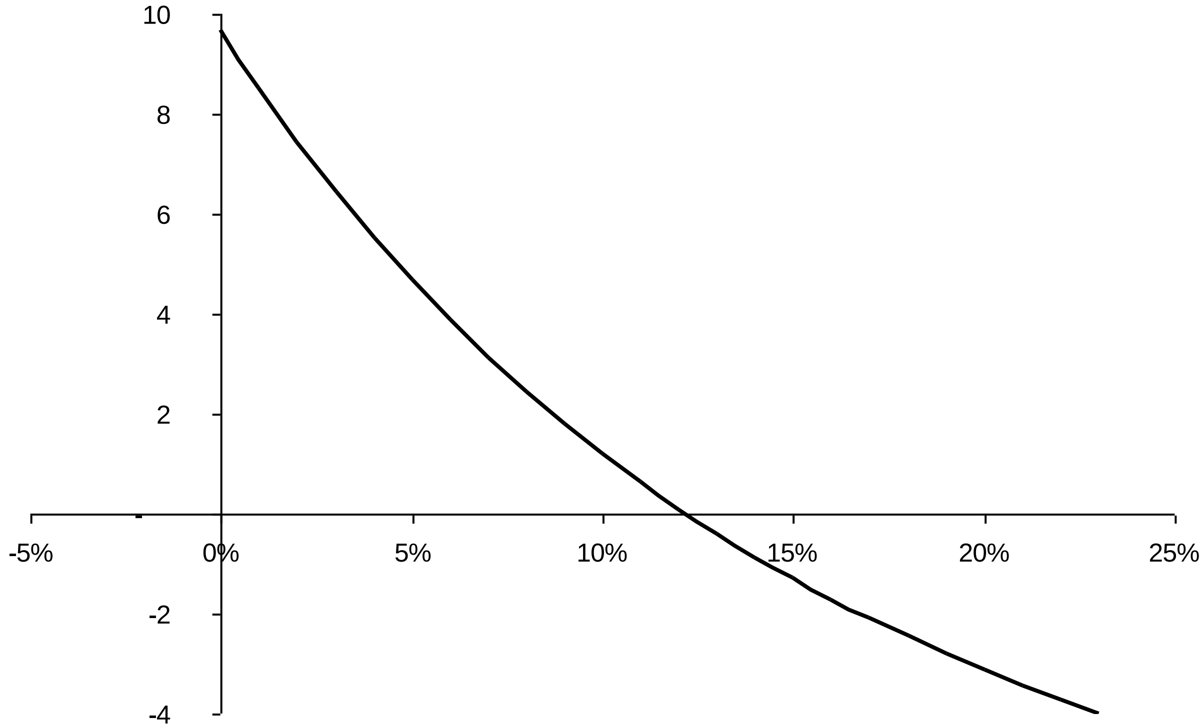
Solving Numerically Using the Newton-Raphson Algorithm

If the equation for f(x) is not known, an alternative method is to estimate the derivative numerically using the formula:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Choosing a sufficiently small enough value for h allows the user to estimate f'(x) and, therefore, $f_1'(x_1)$, $f_2'(x_2)$ and $f_3'(x_3)$ can all be determined thereby solving the problem of differentiating f(x) algebraically.

The following example, which has a root of 12.19%, illustrates the approach:



Some of the values for the above function (which are needed for the ensuing calculations) are contained in the following table:

x	f(x)
0.10000	1.18713
0.10001	1.18654
0.12012	0.08987
0.12013	0.08936
0.12188	0.00051
0.12189	0.00001

Starting with a guess of 10% the resulting NPV is 1.18713 ie. the [x, y] point on the curve f(x) is [0.1, 1.18713]. If we assume h = 0.00001 then our estimate for $f_1'(x_1)$ is:

$$\begin{aligned}&\frac{f(x_1+h)-f(x_1)}{h}\\&=\frac{f(0.1+0.00001)-f(0.1)}{0.00001}\\&=\frac{1.18654-1.18713}{0.00001}\\&=-59.00\end{aligned}$$

We now need to determine the data point $[x_2, 0]$:

$$\begin{aligned}x_2 &= x_1-\frac{f(x_1)}{f'(x_1)}\\ \therefore x_2 &= 0.1-\frac{1.18713}{-59.00}\\ \therefore x_2 &= 0.12012\end{aligned}$$

Using this value, $f(x_2)$ is 0.08987, ie. the point on the curve is [0.12012, 0.08987]. Our estimate for $f_2'(x_2)$ is:

$$\begin{aligned}&\frac{f(x_2+h)-f(x_2)}{h}\\&=\frac{f(0.12012+0.00001)-f(0.12012)}{0.00001}\\&=\frac{0.08936-0.08987}{0.00001}\\&=-51.00\end{aligned}$$

Hence, x_3 is:

$$\begin{aligned}x_3 &= x_2-\frac{f(x_2)}{f'(x_2)}\\ \therefore x_3 &= 0.12012-\frac{0.08987}{-51.00}\\ \therefore x_3 &= 0.12188\end{aligned}$$

Again, f(0.12188) is 0.00051 so $f_3'(0.12188)$ is:

$$\begin{aligned}&\frac{f(x_3+h)-f(x_3)}{h}\\&=\frac{f(0.12188+0.00001)-f(0.12188)}{0.00001}\\&=\frac{0.00001-0.00051}{0.00001}\\&=-50.00\end{aligned}$$

And x_4 is:

$$x_4 = x_3 - \frac{f(x_3)}{f'(x_3)}$$

$$\therefore x_4 = 0.12188 - \frac{0.00051}{-50.00}$$

$$\therefore x_4 = 0.12189$$

Which is only a 0.00001 change from the previous iteration, ie. we are now accurate to 4 decimal places. All this in only 4 iterations! In fact, if we had chosen our beginning value (x_1) as 0 instead of 0.1, the number of iterations required to achieve the same accuracy would have been only 6. If we had chosen x_1 =1 instead of 0.1, the number of iterations would have been 7.

The above iterative process can be modelled within a spreadsheet such as Microsoft Excel or OpenOffice Calc using data tables referencing the cell containing the NPV value. However, care should be taken to ensure the data tables are accurate by repeatedly recalculating (pressing the F9 key in Excel) until there is no visible change. A better solution is to codify the above algorithm (eg. using Excel's VBA) thus avoiding the F9-update problem.

To simplify, the algorithm to be coded is:

Let g = guess and let counter = 0
Loop while c is less than the desired maximum number of iterations, eg. loop while c < 10
    Let x = g – f(g) / f'(g)
    If the absolute value of f(x) is less than the required accuracy, then exit the loop
    Let g take on x's value in readiness for the next cycle of the loop
    Increment counter by 1
    Return to start of the loop for the next iteration

Example code for three applications of the Newton-Raphson algorithm follow.

Application 1: Internal Rate Of Return

The following Excel VBA code illustrates Newton-Raphson using Excel's built-in NPV function to find the IRR:

```
'// Make function visible to whole project
Public Function InternalRateOfReturn( _
ByRef rngCashflows As Range, _
ByRef dblGuess As Double) As Double

'// Declare local variables
Dim intCount As Integer
Dim dblPrecision As Double
Dim dblNextGuess As Double
Dim dblFunction As Double
Dim dblH As Double
Dim dblFunctionH As Double
Dim dblFunctionDeriv As Double

'// Initialise variables
intCount = 0
dblPrecision = 0.00001
dblH = 0.0000000001
dblNextGuess = dblGuess

'// Perform Newton-Raphson algorithm
Do While intCount < 10
    dblFunction = Application.WorksheetFunction.NPV(dblNextGuess, rngCashflows)
    If Abs(dblFunction) < dblPrecision Then
        Exit Do
    End If
    dblFunctionH = Application.WorksheetFunction.NPV(dblNextGuess + dblH, rngCashflows)
    dblFunctionDeriv = (dblFunctionH - dblFunction) / dblH
    dblNextGuess = dblNextGuess - dblFunction / dblFunctionDeriv
    intCount = intCount + 1
Loop

'// Return value
InternalRateOfReturn = dblNextGuess
End Function
```

Application 2: Implied Volatility

The following code uses the Newton-Raphson algorithm to find the volatility implied ("ImpliedVolatility") by the Black-Scholes option pricing model ("BlackScholes") from an observed option price:

```
Public Function ImpliedVolatility( _
ByRef dblPrice As Double, _
ByRef dteValue As Date, _
ByRef dblSpot As Double, _
ByRef dblStrike As Double, _
ByRef dteExpiry As Date, _
ByRef dblRate As Double, _
ByRef strType As String) As Double

'// Declare local variables
Dim intCount As Integer
Dim dblAccuracy As Double
Dim dblNextGuess As Double
Dim dblFunction As Double
Dim dblH As Double
Dim dblFunctionH As Double
Dim dblFirstDerivative As Double

'// Initialise variables
intCount = 0
dblAccuracy = 0.00001
dblH = 0.0000000001
dblNextGuess = 0.2

'// Newton-Raphson Algorithm
Do While intCount < 10
    dblFunction = GetDiff("Premium", dteValue, dblSpot, dblStrike, dblNextGuess, dteExpiry, dblRate, strType, dblPrice)
    If Abs(dblFunction) < dblAccuracy Then Exit Do
    dblFunctionH = GetDiff("Premium", dteValue, dblSpot, dblStrike, dblNextGuess + dblH, dteExpiry, dblRate, strType, dblPrice)
    dblFirstDerivative = (dblFunctionH - dblFunction) / dblH
    dblNextGuess = dblNextGuess - dblFunction / dblFirstDerivative
    intCount = intCount + 1
Loop

'// Return value
ImpliedVolatility = dblNextGuess
End Function


Public Function GetDiff( _
ByRef strCalc As String, _
ByRef dteValue As Date, _
ByRef dblSpot As Double, _
ByRef dblStrike As Double, _
ByRef dblVol As Double, _
ByRef dteExpiry As Date, _
ByRef dblRate As Double, _
ByRef strType As String, _
ByRef dblPrice As Double) As Double

'// Declare variables
Dim dblCalculatedPremium As Double
Dim dblObservedPrice As Double
Dim dblDifference As Double

'// Calculate difference between the theoretical (calculated) option premium and the actual (observed) price
dblCalculatedPremium = BlackScholes(strCalc, dteValue, dblSpot, dblStrike, dblVol, dteExpiry, dblRate, strType)
dblObservedPrice = dblPrice
dblDifference = dblCalculatedPremium - dblObservedPrice

'// Return value
GetDiff = dblDifference
End Function
```

```
Public Function BlackScholes( _
ByRef strCalc As String, _
ByRef dteValue As Date, _
ByRef dblSpot As Double, _
ByRef dblStrike As Double, _
ByRef dblVol As Double, _
ByRef dteExpiry As Date, _
ByRef dblRate As Double, _
ByRef strType As String) As Double

'// Initialise variables
Dim dblTerm As Double
Dim dblD1 As Double
Dim dblD2 As Double
Dim dblN1 As Double
Dim dblN2 As Double
Dim dblPremium As Double
Dim dblDelta As Double
Dim dblGamma As Double
Dim dblVega As Double
Dim dblTheta As Double
Dim dblRho As Double
Dim dblPDF As Double
Dim dblThetaCommon As Double
Dim dblRhoCommon As Double

'// Calculate re-useable variables
dblTerm = (dteExpiry - dteValue) / 365
dblD1 = (Log(dblSpot / dblStrike) + (dblRate + 0.5 * dblVol ^ 2) * dblTerm) / (dblVol * Sqr(dblTerm))
dblD2 = dblD1 - dblVol * Sqr(dblTerm)
dblN1 = Application.WorksheetFunction.NormSDist(dblD1)
dblN2 = Application.WorksheetFunction.NormSDist(dblD2)
dblPDF = Exp(-0.5 * dblD1 ^ 2) / Sqr(2 * Application.WorksheetFunction.Pi)
dblGamma = dblPDF / (dblSpot * dblVol * Sqr(dblTerm))
dblVega = dblPDF * dblSpot * Sqr(dblTerm)
dblThetaCommon = dblSpot * dblPDF * dblVol / (2 * Sqr(dblTerm))
dblRhoCommon = dblStrike * Exp(-dblRate * dblTerm) * dblTerm

'// Calculate Price (Premium) and the Greeks
Select Case strType
    Case "Call"
        dblPremium = dblSpot * dblN1 - dblStrike * Exp(-dblRate * dblTerm) * dblN2
        dblDelta = dblN1
        dblTheta = dblThetaCommon + dblStrike * Exp(-dblRate * dblTerm) * dblRate * (dblN2)
        dblRho = dblRhoCommon * (dblN2)
    Case "Put"
        dblPremium = -dblSpot * (1 - dblN1) + dblStrike * Exp(-dblRate * dblTerm) * (1 - dblN2)
        dblDelta = dblN1 - 1
        dblTheta = dblThetaCommon + dblStrike * Exp(-dblRate * dblTerm) * dblRate * (dblN2 - 1)
        dblRho = dblRhoCommon * (dblN2 - 1)
End Select

'// Return value
Select Case strCalc
    Case "Premium"
        BlackScholes = dblPremium
    Case "Delta"
        BlackScholes = dblDelta
    Case "Gamma"
        BlackScholes = dblGamma
    Case "Vega"
        BlackScholes = dblVega
    Case "Theta"
        BlackScholes = dblTheta
    Case "Rho"
        BlackScholes = dblRho
End Select
End Function
```

Application 3: Fixed Rate of an Interest Rate Swap

Given a set of dates, discount factors, a notional profile and the floating leg margin for an interest rate swap, the Newton-Raphson algorithm can be used to determine the implied fixed rate:

```
'// Make the function visible to the whole project
Public Function SwapRate( _
ByRef rngDates As Range, _
ByRef rngDFs As Range, _
ByRef rngNotional As Range, _
ByRef dblMargin) As Double

'// Declare variables
Dim intCount As Integer
Dim dblAccuracy As Double
Dim dblH As Double
Dim dblNextGuess As Double
Dim dblFunction As Double
Dim dblFunctionH As Double
Dim dblFirstDerivative As Double

'// Initialise variables
intCount = 0
dblAccuracy = 0.00001
dblH = 0.0000000001
dblNextGuess = 0.05

'// Newton-Raphson Algorithm
Do While intCount < 10
    dblFunction = GetNpvOfSwap(rngDates, rngDFs, rngNotional, dblNextGuess, dblMargin)
    If Abs(dblFunction) < dblAccuracy Then Exit Do
    dblFunctionH = GetNpvOfSwap(rngDates, rngDFs, rngNotional, dblNextGuess + dblH, dblMargin)
    dblFirstDerivative = (dblFunctionH - dblFunction) / dblH
    dblNextGuess = dblNextGuess - dblFunction / dblFirstDerivative
    intCount = intCount + 1
Loop

'// Return value
SwapRate = dblNextGuess
End Function


Private Function GetNpvOfSwap( _
ByRef rngDates As Range, _
ByRef rngDFs As Range, _
ByRef rngNotional As Range, _
ByRef dblFixedRate As Double, _
ByRef dblMargin) As Double

'// Declare variables
Dim dblPvOfFixedLeg As Double
Dim dblPvOfFloatingLeg As Double
Dim dblNpvOfSwap As Double

'// Calculate PV of legs and NPV of swap
dblPvOfFixedLeg = GetPvOfFixedLeg(rngDates, rngDFs, rngNotional, dblFixedRate)
dblPvOfFloatingLeg = GetPvOfFloatingLeg(rngDates, rngDFs, rngNotional, dblMargin)
dblNpvOfSwap = dblPvOfFixedLeg - dblPvOfFloatingLeg

'// Return value
GetNpvOfSwap = dblNpvOfSwap
End Function


Private Function GetPvOfFixedLeg( _
ByRef rngDates As Range, _
ByRef rngDFs As Range, _
ByRef rngNotional As Range, _
ByRef dblFixedRate As Double) As Double
```

```
'// Declare variables
Dim lngLoop As Long
Dim dblFixedInterest As Double
Dim dblPvOfFixedInterest As Double

'// Initialise variables
lngLoop = 0
dblFixedInterest = 0
dblPvOfFixedInterest = 0

'// Loop through each date but one
For lngLoop = 2 To rngDates.Cells.Count
        dblFixedInterest = rngNotional.Cells(lngLoop - 1) * dblFixedRate * (rngDates.Cells(lngLoop) - rngDates.Cells(lngLoop - 1)) / 365
        dblPvOfFixedInterest = dblPvOfFixedInterest + dblFixedInterest * rngDFs.Cells(lngLoop)
Next lngLoop

'// Return the value
GetPvOfFixedLeg = dblPvOfFixedInterest
End Function


Private Function GetPvOfFloatingLeg( _
ByRef rngDates As Range, _
ByRef rngDFs As Range, _
ByRef rngNotional As Range, _
ByRef dblMargin) As Double

'// Declare variables
Dim lngLoop As Long
Dim dblNumDaysInRoll As Double
Dim dblYearFraction As Double
Dim dblFloatingRate As Double
Dim dblFloatingInterest As Double
Dim dblPvOfFloatingInterest As Double

'// Initialise variables
lngLoop = 0
dblNumDaysInRoll = 0
dblYearFraction = 0
dblFloatingInterest = 0
dblPvOfFloatingInterest = 0

'// Loop through each date but one
For lngLoop = 2 To rngDates.Cells.Count
        dblNumDaysInRoll = rngDates.Cells(lngLoop) - rngDates.Cells(lngLoop - 1)
        dblYearFraction = dblNumDaysInRoll / 365
        dblFloatingRate = (rngDFs.Cells(lngLoop - 1) / rngDFs.Cells(lngLoop) - 1) / dblYearFraction + dblMargin
        dblFloatingInterest = rngNotional.Cells(lngLoop - 1) * dblFloatingRate * dblYearFraction
        dblPvOfFloatingInterest = dblPvOfFloatingInterest + dblFloatingInterest * rngDFs.Cells(lngLoop)
Next lngLoop

'// Return the value
GetPvOfFloatingLeg = dblPvOfFloatingInterest
End Function
```

Application 4: Floating Rate Margin of an Interest Rate Swap

Given a set of dates, discount factors, notional profile and the fixed rate for a swap, the Newton-Raphson algorithm can be used to determine the margin on the floating leg of the swap:

```
'// Make the function visible to the whole project
Public Function SwapMargin( _
ByRef rngDates As Range, _
ByRef rngDFs As Range, _
ByRef rngNotional As Range, _
ByRef dblFixedRate As Double) As Double

'// Declare variables
Dim intCount As Integer
Dim dblAccuracy As Double
Dim dblH As Double
Dim dblNextGuess As Double
Dim dblFunction As Double
Dim dblFunctionH As Double
Dim dblFirstDerivative As Double

'// Initialise variables
intCount = 0
dblAccuracy = 0.00001
dblH = 0.0000000001
dblNextGuess = 0.005

'// Newton-Raphson Algorithm
Do While intCount < 10
    dblFunction = GetNpvOfSwap(rngDates, rngDFs, rngNotional, dblFixedRate, dblNextGuess)
    If Abs(dblFunction) < dblAccuracy Then Exit Do
    dblFunctionH = GetNpvOfSwap(rngDates, rngDFs, rngNotional, dblFixedRate, dblNextGuess + dblH)
    dblFirstDerivative = (dblFunctionH - dblFunction) / dblH
    dblNextGuess = dblNextGuess - dblFunction / dblFirstDerivative
    intCount = intCount + 1
Loop

'// Return value
SwapMargin = dblNextGuess
End Function
```

NB: the above Floating Rate Margin code requires access to the private functions specified in the previous Fixed Rate code.