```
Open "numerical_roots_output5.txt" For Output As #1

Dim Shared FORCE_IVT%   ' Should check Intermediate Value Theorem
FORCE_IVT% = 1

Dim Shared DEFAULT_NUMERICAL_DERIVATIVE_STEP#
DEFAULT_NUMERICAL_DERIVATIVE_STEP# = 1E-7

MAX_ITERATIONS% = 100
TOLERANCE# = 1E-5   ' Error that can be tolerated

x0# = 0   ' First initial approximation
x1# = 1   ' Second initial approximation

pl "..............Numerical root finding methods ................"
lb
pl "## 1. Binary Bisection Method"
root_bisect# = bisect#(x0#, x1#, MAX_ITERATIONS%, TOLERANCE#)
pl "Bisection Root: " + trim_val$(root_bisect#)
lb
pl "## 2. Regular Falsi (Chord) Method"
root_chord# = chord#(x0#, x1#, MAX_ITERATIONS%, TOLERANCE#)
pl "Chord Root: " + trim_val$(root_chord#)
lb
pl "## 3. Secant Method"
root_secant# = secant#(x0#, x1#, MAX_ITERATIONS%, TOLERANCE#)
pl "Secant Root: " + trim_val$(root_secant#)
lb
pl "## 4. Newton-Raphson Method"
root_raphson# = raphson#(x0#, MAX_ITERATIONS%, TOLERANCE#)
pl "Raphson Root: " + trim_val$(root_raphson#)

' Function Definitions
Function f1# (x#)
  f1# = x# - (3 ^ 0.5)
End Function

Function f2# (x#)
  f2# = (x# * x# * x#) - (x# * x#) + 1
End Function

Function f3# (x#)
  f3# = (x# * Exp(x#)) - Cos(x#)
End Function

Function f4# (x#)
  f4# = (x# * x# * x#) - Sin(x#) + 4
End Function

Function f5# (x#)
  f5# = (x# * Exp(x#)) - 1
End Function

' Function used all over the program
' Change the base function here
Function f# (x#)
  f# = f5#(x#)
End Function

Function numericalDerivative# (x#, h#)
  numericalDerivative# = (f#(x# + h#) - f#(x#)) / h#
End Function

Function numericalDerivativeDef# (x#)
  numericalDerivativeDef# = numericalDerivative#(x#, DEFAULT_NUMERICAL_DERIVATIVE_STEP#)
End Function

Function isBw% (v#, left#, right#)  ' checks if a number is in between left and right values
  If (v# >= left# And v# <= right#) Or (v# <= left# And v# >= right#) Then
    isBw% = 1
  Else
    isBw% = 0
  End If
End Function

Function bisect# (firstInitialApproximation#, secondInitialApproximation#, maxItrs%, tolerance#)
  left# = firstInitialApproximation#
  right# = secondInitialApproximation#
  leftVal# = f#(left#)
  rightVal# = f#(right#)

  If FORCE_IVT% = 1 And isBw%(0, leftVal#, rightVal#) = 0 Then
    pl "Bisection -> FATAL: IVT violated due to bad initial approximation  x0: " + trim_val$(left#) + ",  x1: " +
trim_val$(right#)
    bisect# = -1 ' Invalid initial approximations
  Else
    mid# = -1
    midVal# = -1
    itr% = 1

    Do
      mid# = (left# + right#) / 2
      midVal# = f#(mid#)
      pl "Bisection Iteration " + trim_val$(itr%) + " -> Root: " + trim_val$(mid#) + ",  Root Value: " +
trim_val$(midVal#)

      If Abs(midVal#) <= tolerance# Then
        pl "Bisection -> Root Found at iteration " + trim_val$(itr%)
        Exit Do
      End If

      If isBw%(0, leftVal#, midVal#) = 1 Then
        right# = mid#
      ElseIf isBw%(0, midVal#, rightVal#) = 1 Then
        left# = mid#
      ElseIf FORCE_IVT% = 1 Then
        pl "Bisection -> FATAL: IVT violated, returning last root..."
        Exit Do
      End If

      itr% = itr% + 1
    Loop While itr% <= maxItrs%

    bisect# = mid#
  End If
End Function

Function chord# (firstInitialApproximation#, secondInitialApproximation#, maxItrs%, tolerance#)
  left# = firstInitialApproximation#
  right# = secondInitialApproximation#
```

```
  leftVal# = f#(left#)
  rightVal# = f#(right#)

  If FORCE_IVT% = 1 And isBw%(0, leftVal#, rightVal#) = 0 Then
    pl "Chord -> FATAL: IVT violated due to bad initial approximation  x0: " + trim_val$(left#) + ",  x1: " +
trim_val$(right#)
    chord# = -1 ' Invalid initial approximations
  Else
    mid# = -1
    midVal# = -1
    itr% = 1

    Do
      mid# = ((right# * leftVal#) - (left# * rightVal#)) / (leftVal# - rightVal#)
      midVal# = f#(mid#)
      pl "Chord Iteration " + trim_val$(itr%) + " -> Root: " + trim_val$(mid#) + ",  Root Value: " + trim_val$(midVal#)

      If Abs(midVal#) <= tolerance# Then
        pl "Chord -> Root found at iteration " + trim_val$(itr%)
        Exit Do
      End If

      If isBw%(0, leftVal#, midVal#) = 1 Then
        right# = mid#
      ElseIf isBw%(0, midVal#, rightVal#) = 1 Then
        left# = mid#
      ElseIf FORCE_IVT% = 1 Then
        pl "Chord -> FATAL: IVT violated, returning last root..."
        Exit Do
      End If

      itr% = itr% + 1
    Loop While itr% <= maxItrs%

    chord# = mid#
  End If
End Function

Function secant# (firstInitialApproximation#, secondInitialApproximation#, maxItrs%, tolerance#)
  left# = firstInitialApproximation#
  right# = secondInitialApproximation#
  leftVal# = f#(left#)
  rightVal# = f#(right#)

  If FORCE_IVT% = 1 And isBw%(0, leftVal#, rightVal#) = 0 Then
    pl "Secant -> FATAL: IVT violated due to bad initial approximation  x0: " + trim_val$(left#) + ",  x1: " +
trim_val$(right#)
    secant# = -1 ' Invalid initial approximations
  Else
    mid# = -1
    midVal# = -1
    itr% = 1

    Do
      mid# = ((right# * leftVal#) - (left# * rightVal#)) / (leftVal# - rightVal#)
      midVal# = f#(mid#)
      pl "Secant Iteration " + trim_val$(itr%) + " -> Root: " + trim_val$(mid#) + ",  Root Value: " + trim_val$(midVal#)

      If Abs(midVal#) <= tolerance# Then
        pl "Secant -> Root Found at iteration " + trim_val$(itr%)
        Exit Do
      End If

      left# = right#
      right# = mid#

      itr% = itr% + 1
    Loop While itr% <= maxItrs%

    secant# = mid#
  End If
End Function

Function raphson# (initialApproximation#, maxItrs%, tolerance#)
  cur# = initialApproximation#
  curVal# = -1
  itr% = 0

  Do
    curVal# = f#(cur#)
    pl "Raphson Iteration " + trim_val$(itr%) + " -> Root: " + trim_val$(cur#) + ",  Root Value: " + trim_val$(curVal#)

    If Abs(curVal#) <= tolerance# Then
      pl "Raphson -> Root Found at iteration " + trim_val$(itr%)
      Exit Do
    End If

    derVal# = numericalDerivativeDef#(cur#)
    If derVal# = 0 Then
      pl "Raphson -> FATAL: Derivative is 0 at current root. Terminating..."
      Exit Do
    End If

    cur# = cur# - (curVal# / derVal#)
    itr% = itr% + 1
  Loop While itr% <= maxItrs%

  raphson# = cur#
End Function

' ..... Formatting .....
Sub p (s$)  ' Print and log a given string WITHOUT line break
  Print s$;
  Print #1, s$;
End Sub

Sub lb  ' Print and log a line break
  Print
  Print #1, ""
End Sub

Sub pl (s$)  ' Print and log given string WITH line break
  Print s$
  Print #1, s$
End Sub

Function trim_val$ (i#)
  trim_val$ = LTrim$(RTrim$(Str$(i#)))
End Function
```