Examples

Example 1: Log Gamma Function

```
Option Explicit
Option Base 1

' function declaration, as supplied in VB6 headers
Declare Function S14ABF Lib "FLDLL254M_nag.dll" ( _
  ByRef x As Double, _
  ByRef ifail As Long _
) As Double

Function NAG_logGamma(x As Variant) As Variant
  ' return the log of the gamma function

  ' returned value is a variant so we can return either the
  ' calculated value or an error message
  Dim RX As Double
  Dim ifail As Long

  ' convert the supplied input to a scalar double
  RX = NAG_GetDoubleScalar(x)

  ' set the NAG error mechanism to a quiet soft exit
  ifail = 1

  ' call the NAG routine
  NAG_logGamma = S14ABF(RX, ifail)

  ' handle any errors
  If (ifail <> 0) Then
    NAG_logGamma = "S14ABF returned with IFAIL = " + CStr(ifail)
  End If
End Function
```

Example 2: Summary Statistics

```
Option Explicit
Option Base 1

' subroutine declaration, as supplied in VB6 headers
Declare Sub G01AAF Lib "FLDLL254M_nag.dll" ( _
  ByRef n As Long, _
  ByRef x As Double, _
  ByRef iwt As Long, _
  ByRef wt As Double, _
  ByRef xmean As Double, _
  ByRef s2 As Double, _
  ByRef s3 As Double, _
  ByRef s4 As Double, _
  ByRef xmin As Double, _
  ByRef xmax As Double, _
  ByRef wtsum As Double, _
  ByRef ifail As Long _
)

Function NAG_summaryStats(RX As Range, Optional RWT As Range) _
         As Variant
  ' produce some (optionally weighted) summary statistics
  Dim n As Long, iwt As Long, ifail As Long
  Dim lenx As Long, lenwt As Long
  Dim xmean As Double, s2 As Double, s3 As Double, s4 As Double
  Dim xmin As Double, xmax As Double, wtsum As Double
  Dim x() As Double, wt() As Double
  Dim vntOutput As Variant

  ' extract the data
  Call NAG_GetDoubleVectorFromRange(RX, lenx, x)
  Call NAG_GetDoubleVectorFromRange(RWT, lenwt, wt)

  n = lenx
  If (lenwt <> n) Then
    ' assume unweighted analysis
    iwt = 0
    ReDim wt(n)
  Else
    iwt = 1
  End If

  ' set the NAG error mechanism to a quiet soft exit
  ifail = 1

  ' calculate some summary statistics
  ' NB: The first element is passed to array arguments rather
  ' than the array itself, i.e. x(1) rather than x
  Call G01AAF(n, x(1), iwt, wt(1), xmean, s2, s3, s4, xmin, _
              xmax, wtsum, ifail)
```

```
  If (ifail <> 0 And ifail <> 2) Then
    ' handle any errors, IFAIL = 2 is a warning and useful
    ' information is returned
    ReDim vntOutput(1, 1)
    vntOutput(1, 1) = "G01AAF returned with IFAIL = " + CStr(ifail)

  Else
    ' no errors (or IFAIL = 2, in which case all output,
    ' except S2, S3 and S4 is returned)
    ReDim vntOutput(8, 2)

    vntOutput(1, 1) = "Number of valid observations"
    vntOutput(1, 2) = iwt
    vntOutput(2, 1) = "Mean"
    vntOutput(2, 2) = xmean
    vntOutput(3, 1) = "Standard Deviation"
    vntOutput(3, 2) = IIf(ifail = 2, "NA", s2)
    vntOutput(4, 1) = "Skewness"
    vntOutput(4, 2) = IIf(ifail = 2, "NA", s3)
    vntOutput(5, 1) = "Kurtosis"
    vntOutput(5, 2) = IIf(ifail = 2, "NA", s4)
    vntOutput(6, 1) = "Minimum"
    vntOutput(6, 2) = xmin
    vntOutput(7, 1) = "Maximum"
    vntOutput(7, 2) = xmax
    vntOutput(8, 1) = "Sum of Weights"
    vntOutput(8, 2) = wtsum
  End If

  NAG_summaryStats = vntOutput
End Function
```

Example 3: Modified Bessel Function

```
Option Explicit
Option Base 1

' function declaration, as supplied in VB6 headers
Declare Sub S18DCF Lib "FLDLL254M_nag.dll" ( _
  ByRef fnu As Double, _
  ByRef z As Complex, _
  ByRef n As Long, _
  ByVal scal As String, ByVal scallength As Long, _
  ByRef cy As Complex, _
  ByRef nz As Long, _
  ByRef ifail As Long _
)
' complex defined type, as supplied in the VB6 headers
Type Complex
  Real_Part As Double
  Imag_Part As Double
End Type

Function NAG_besselK(x As Variant, nu As Variant, _
          Optional scaled As Boolean = False) As Variant
  ' return the modified Bessel function, K_v(x), for real x and v
  Dim cy(1) As Complex, z As Complex
  Dim RX As Double, rnu As Double
  Dim n As Long, ifail As Long, nz As Long, scallength As Long
  Dim scal As String

  ' convert the supplied input to a scalar doubles
  RX = NAG_GetDoubleScalar(x)
  rnu = NAG_GetDoubleScalar(nu)

  ' populate the Complex type
  z.Real_Part = RX
  z.Imag_Part = 0#

  ' set up the scaled flag
  scal = IIf(scaled, "S", "U")
  scallength = 1

  ' we only want a single evaluation at nu
  n = 1

  ' set the NAG error mechanism to a quiet soft exit
  ifail = 1

  ' get the modified Bessel function K_v, where v can be non-integer
  Call S18DCF(rnu, z, n, scal, scallength, cy(1), nz, ifail)

  If (ifail <> 0) Then
    ' handle any errors
    NAG_besselK = "S14ABF returned with IFAIL = " + CStr(ifail)

  Else
    ' no errors, so return the real part of the result
    NAG_besselK = cy(1).Real_Part
  End If
End Function
```

Example 4: System Non-Linear Equations

```
' function declaration, as supplied in VB6 headers
Declare Function X02AJF Lib "FLDLL254M_nag.dll" () As Double
Declare Sub C05QBF Lib "FLDLL254M_nag.dll" ( _
  ByVal fcn As Long, _
  ByRef n As Long, _
  ByRef x As Double, _
  ByRef fvec As Double, _
  ByRef xtol As Double, _
  ByRef iuser As Long, _
  ByRef ruser As Double, _
  ByRef ifail As Long _
)

' Copies memory from pointer
' NB: that hpvDest is byRef and hpvSource is ByVal
' this is the opposite to CopyMemToPtr
Declare Sub CopyMemFromPtr Lib "kernel32" Alias "RtlMoveMemory" ( _
  ByRef hpvDest As Any, ByVal hpvSource As Any, _
  ByVal cbCopy As Long)

' Copies memory to pointer
' NB: that hpvDest is byVal and hpvSource is ByRef
' this is the opposite to CopyMemFromPtr
Declare Sub CopyMemToPtr Lib "kernel32" Alias "RtlMoveMemory" ( _
  ByVal hpvDest As Long, ByRef hpvSource As Any, _
  ByVal cbCopy As Long)

Function NAG_systemNonLinear(rx As Range, ru As Range) As Variant
  ' solve a system of non-linear equations
  Dim xtol As Double
  Dim ifail As Long, n As Long, lenu As Long
  Dim i As Long
  Dim fvec() As Double, ruser() As Double, x() As Double
  Dim iuser(1) As Long
  Dim vntOutput As Variant

  ' extract the data
  Call NAG_GetDoubleVectorFromRange(rx, n, x)
  Call NAG_GetDoubleVectorFromRange(ru, lenu, ruser)

  ' set the tolerance as the square-root of machine precision
  xtol = Sqr(X02AJF())

  ' allocate some memory for the output
  ReDim fvec(n)

  ' set the NAG error mechanism to a quiet soft exit
  ifail = 1

  ' call the NAG routine
  Call C05QBF(AddressOf Example4_UserFun, n, x(1), fvec(1), xtol, _
              iuser(1), ruser(1), ifail)

  If (ifail <> 0) Then
    ' handle any errors
    ReDim vntOutput(1, 1)
    vntOutput(1, 1) = "C05QBF returned with IFAIL = " + CStr(ifail)
```

```
  Else
    ' return the results
    ReDim vntOutput(n, 2)
    For i = 1 To n
      vntOutput(i, 1) = x(i)
      vntOutput(i, 2) = fvec(i)
    Next i
  End If

  NAG_systemNonLinear = vntOutput
End Function

Sub Example4_UserFun( _
    ByRef n As Long, _
    ByVal x_rptr As Long, _
    ByVal fvec_rptr As Long, _
    ByVal iuser_iptr As Long, _
    ByVal ruser_rptr As Long, _
    ByRef iflag As Long _
    )
  ' function of tridiagonal equations

  Dim x() As Double, fvec() As Double, ruser() As Double
  Dim lruser As Long, i As Long

  ' we are using 3 elements of RUSER
  lruser = 3
  ReDim x(n), fvec(n), ruser(lruser)

  ' copy array input from x_rptr and ruser_rptr into local arrays
  ' we are not using iuser_rptr
  Call CopyMemFromPtr(x(1), x_rptr, n * Len(x(1)))
  Call CopyMemFromPtr(ruser(1), ruser_rptr, lruser * Len(ruser(1)))

  ' evaluate the function of interest
  fvec(1) = (ruser(1) - ruser(2) * x(1)) * x(1) - _
            ruser(3) * x(2) + 1#
  For i = 2 To n - 1
    fvec(i) = -x(i - 1) + (ruser(1) - ruser(2) * x(i)) * x(i) - _
              ruser(3) * x(i + 1) + 1#
  Next i
  fvec(n) = -x(n - 1) + (ruser(1) - ruser(2) * x(n)) * x(n) + 1#

  ' copy results from fvec to fvec_rptr
  Call CopyMemToPtr(fvec_rptr, fvec(1), n * Len(fvec(1)))

End Sub
```