THE ORIGINAL VBA FUNCTIONS from Excel

```
Const Log2# = 0.693147180559945
Const twoPI# = 6.28318530717959

Function vaBermPutCONV(S#, K#, r#, q#, Tyr#, vol#, nStepT&, nPower2&) As Double
    Dim dT#, erdT#, vol2#, rnmudT#, SqrV#, dy#, ydisc#, yeps#, du#, crfi#, cifi#
    Dim xdisc#, xeps#, xShift#, crecfi#, ciecfi#
    Dim delta&, i&, iBase&, id&, iSwitch&, j&
    Dim a1vec
    Dim c1vec

    Dim bgvec() As Double
    Dim ciavec() As Double
    Dim cicfvec() As Double
    Dim cilncfvec() As Double
    Dim ciaecfvec() As Double
    Dim cravec() As Double
    Dim crcfvec() As Double
    Dim crlncfvec() As Double
    Dim craecfvec() As Double
    Dim cvec() As Double
    Dim evec() As Double
    Dim uvec() As Double
    Dim vvec() As Double
    Dim vwtwvec() As Double
    Dim wtwvec() As Double
    Dim wvec() As Double
    Dim xvec() As Double
    Dim yvec() As Double
    Dim zerovec() As Double
    ReDim bgvec(nPower2 - 1)
    ReDim ciavec(nPower2 - 1)
    ReDim cicfvec(nPower2 - 1)
    ReDim cilncfvec(nPower2 - 1)
    ReDim ciaecfvec(nPower2 - 1)
    ReDim cravec(nPower2 - 1)
    ReDim crcfvec(nPower2 - 1)
    ReDim crlncfvec(nPower2 - 1)
    ReDim craecfvec(nPower2 - 1)
    ReDim cvec(nPower2 - 1)
    ReDim evec(nPower2 - 1)
    ReDim uvec(nPower2 - 1)
    ReDim vvec(nPower2 - 1)
    ReDim vwtwvec(nPower2 - 1)
    ReDim wtwvec(nPower2 - 1)
    ReDim wvec(nPower2 - 1)
    ReDim xvec(nPower2 - 1)
    ReDim yvec(nPower2 - 1)
    ReDim zerovec(nPower2 - 1)

    iBase = 0
    iSwitch = 1
    delta = 20
    dT = Tyr / nStepT
    erdT = Exp(-r * dT)
    vol2 = vol * vol
    rnmudT = (r - q - 0.5 * vol2) * dT
    SqrV = vol * Sqr(Tyr)
    dy = delta * SqrV / nPower2
    ydisc = Log(K / S)
    yeps = ydisc - vaCeiling(ydisc / dy) * dy
    du = twoPI / (nPower2 * dy)

    For i = 0 To nPower2 - 2 Step 2
        wvec(i) = 1
        wvec(i + 1) = -1
        wtwvec(i) = 1
        wtwvec(i + 1) = -1
    Next i
    wtwvec(0) = 0.5
    wtwvec(nPower2 - 1) = -0.5

    bgvec(0) = -0.5 * nPower2 * dy
    yvec(0) = bgvec(0) + yeps
    uvec(0) = -0.5 * nPower2 * du
    For i = 1 To nPower2 - 1
        bgvec(i) = bgvec(i - 1) + dy
        yvec(i) = bgvec(i) + yeps
        uvec(i) = uvec(i - 1) + du
    Next i
    For i = 0 To nPower2 - 1
        crfi = -0.5 * vol2 * dT * uvec(i) * uvec(i)
        cifi = rnmudT * uvec(i)
        crcfvec(i) = Exp(crfi) * Cos(cifi)
        cicfvec(i) = Exp(crfi) * Sin(cifi)
        evec(i) = Max(K - S * Exp(yvec(i)), 0)
        vvec(i) = evec(i)
        vwtwvec(i) = vvec(i) * wtwvec(i)
        zerovec(i) = 0
    Next i

    For j = nStepT - 1 To 1 Step -1

        a1vec = vaCONVfvec(1, vwtwvec, zerovec, nPower2, iSwitch, iBase)

        For i = 0 To nPower2 - 1
            cravec(i) = a1vec(i, 0) * crcfvec(i) - a1vec(i, 1) * cicfvec(i)
            ciavec(i) = a1vec(i, 0) * cicfvec(i) + a1vec(i, 1) * crcfvec(i)
        Next i

        c1vec = vaCONVfvec(-1, cravec, ciavec, nPower2, iSwitch, iBase)

        For i = 0 To nPower2 - 1
            cvec(i) = erdT * wvec(i) * c1vec(i, 0)
            If cvec(i) > evec(i) Then Exit For
        Next i

        id = i
        If id = 0 Then id = 1

        xdisc = (evec(id - 1) - cvec(id - 1)) * yvec(id) - (evec(id) - cvec(id)) * yvec(id - 1)
        xdisc = xdisc / (evec(id - 1) - evec(id) + cvec(id) - cvec(id - 1))
        xeps = xdisc - vaCeiling(xdisc / dy) * dy
        xShift = xeps - yeps

        For i = 0 To nPower2 - 1
            xvec(i) = bgvec(i) + xeps
            evec(i) = Max(K - S * Exp(xvec(i)), 0)
            crecfi = Cos(xShift * uvec(i))
            ciecfi = Sin(xShift * uvec(i))
            craecfvec(i) = cravec(i) * crecfi - ciavec(i) * ciecfi
            ciaecfvec(i) = cravec(i) * ciecfi + ciavec(i) * crecfi
        Next i

        c1vec = vaCONVfvec(-1, craecfvec, ciaecfvec, nPower2, iSwitch, iBase)

        For i = 0 To nPower2 - 1
            cvec(i) = erdT * wvec(i) * c1vec(i, 0)
            vvec(i) = Max(cvec(i), evec(i))
            vwtwvec(i) = vvec(i) * wtwvec(i)
            yvec(i) = xvec(i)
        Next i

        yeps = xeps

    Next j

    xdisc = 0
    xShift = -yeps
    id = 0.5 * nPower2

    a1vec = vaCONVfvec(1, vwtwvec, zerovec, nPower2, iSwitch, iBase)

    For i = 0 To nPower2 - 1
        crfi = -0.5 * vol2 * dT * uvec(i) * uvec(i)
        cifi = rnmudT * uvec(i)
        crecfi = Exp(crfi) * Cos(cifi + xShift * uvec(i))
        ciecfi = Exp(crfi) * Sin(cifi + xShift * uvec(i))
        craecfvec(i) = a1vec(i, 0) * crecfi - a1vec(i, 1) * ciecfi
        ciaecfvec(i) = a1vec(i, 0) * ciecfi + a1vec(i, 1) * crecfi
    Next i

    c1vec = vaCONVfvec(-1, craecfvec, ciaecfvec, nPower2, iSwitch, iBase)

    vaBermPutCONV = erdT * wvec(id) * c1vec(id, 0)
End Function

Function vaCONVfvec(iFwdInv&, crfvec, cifvec, nPower2&, iSwitch&, iBase&)
    Dim da#, alpha#, beta#, ar#, ai#, tr#, ti#, dar#, fi#
    Dim nBits&
    Dim blockEndL&, blockSizeL&, iL&, indxL&, jL&, jkL&, nL&
    Dim fvec() As Double
    ReDim fvec(nPower2 - 1, 1)

    nBits = Log(nPower2) / Log2

    For iL = 0 To nPower2 - 1
        indxL = iL
        jL = 0
        For jkL = 0 To nBits - 1
            jL = (jL * 2) Or (indxL And 1)
            indxL = vaIntL(indxL, 2)
        Next
        fvec(jL, 0) = crfvec(iL + iBase)
        fvec(jL, 1) = cifvec(iL + iBase)
    Next

    blockEndL = 1
    blockSizeL = 2

    Do While blockSizeL <= nPower2
        da = iFwdInv * twoPI / blockSizeL
        alpha = 2 * Sin(0.5 * da) * Sin(0.5 * da)
        beta = Sin(da)

        iL = 0
        Do While iL < nPower2
            ar = 1
            ai = 0

            jL = iL
            For nL = 0 To blockEndL - 1
                jkL = jL + blockEndL
                tr = ar * fvec(jkL, 0) - ai * fvec(jkL, 1)
                ti = ai * fvec(jkL, 0) + ar * fvec(jkL, 1)
                fvec(jkL, 0) = fvec(jL, 0) - tr
                fvec(jL, 0) = fvec(jL, 0) + tr
                fvec(jkL, 1) = fvec(jL, 1) - ti
                fvec(jL, 1) = fvec(jL, 1) + ti
                dar = alpha * ar + beta * ai
                ai = ai - (alpha * ai - beta * ar)
                ar = ar - dar
                jL = jL + 1
            Next

            iL = iL + blockSizeL
        Loop

        blockEndL = blockSizeL
        blockSizeL = blockSizeL * 2
    Loop

    If iSwitch = 1 Then
        For iL = nPower2 / 2 + 1 To nPower2 - 1
            fi = fvec(iL, 0)
            fvec(iL, 0) = fvec(nPower2 - iL, 0)
            fvec(nPower2 - iL, 0) = fi
            fi = fvec(iL, 1)
            fvec(iL, 1) = fvec(nPower2 - iL, 1)
            fvec(nPower2 - iL, 1) = fi
        Next iL
    End If

    If iFwdInv = -1 Then
        For iL = 0 To nPower2 - 1
            fvec(iL, 0) = fvec(iL, 0) / nPower2
            fvec(iL, 1) = fvec(iL, 1) / nPower2
        Next iL
    End If

    vaCONVfvec = fvec
End Function

Function Max(x1#, x2#) As Double
    If x1 >= x2 Then
        Max = x1
    Else
        Max = x2
    End If
End Function

Function vaCeiling(x#) As Double
    If Int(x) = x Then
        vaCeiling = x
    Else
        vaCeiling = Int(x) + 1
    End If
End Function

Function vaIntL(i1&, i2&) As Long
    vaIntL = Int(i1 / i2)
End Function
```

THE VB.NET FUNCTIONS

```
Const Log2 As Double = 0.693147180559945
Const twoPI As Double = 6.28318530717959

Function vbBermPutCONV(S As Double, K As Double, r As Double, q As Double, Tyr As Double, vol As Double, nStepT As Int32, nPower2 As Int32) As Double
    Dim dT As Double, erdT As Double, vol2 As Double, rnmudT As Double, SqrtV As Double, dy As Double, ydisc As Double, yeps As Double, du As Double, crfi As Double, cifi As Double
    Dim xdisc As Double, xeps As Double, xShift As Double, crecfi As Double, ciecfi As Double
    Dim delta As Int32, i As Int32, iBase As Int32, id As Int32, iSwitch As Int32, j As Int32
    Dim a1vec As Object
    Dim c1vec As Object

    Dim bgvec() As Double
    Dim ciavec() As Double
    Dim cicfvec() As Double
    Dim cilncfvec() As Double
    Dim ciaecfvec() As Double
    Dim cravec() As Double
    Dim crcfvec() As Double
    Dim crlncfvec() As Double
    Dim craecfvec() As Double
    Dim cvec() As Double
    Dim evec() As Double
    Dim uvec() As Double
    Dim vvec() As Double
    Dim vwtwvec() As Double
    Dim wtwvec() As Double
    Dim wvec() As Double
    Dim xvec() As Double
    Dim yvec() As Double
    Dim zerovec() As Double
    ReDim bgvec(nPower2 - 1)
    ReDim ciavec(nPower2 - 1)
    ReDim cicfvec(nPower2 - 1)
    ReDim cilncfvec(nPower2 - 1)
    ReDim ciaecfvec(nPower2 - 1)
    ReDim cravec(nPower2 - 1)
    ReDim crcfvec(nPower2 - 1)
    ReDim crlncfvec(nPower2 - 1)
    ReDim craecfvec(nPower2 - 1)
    ReDim cvec(nPower2 - 1)
    ReDim evec(nPower2 - 1)
    ReDim uvec(nPower2 - 1)
    ReDim vvec(nPower2 - 1)
    ReDim vwtwvec(nPower2 - 1)
    ReDim wtwvec(nPower2 - 1)
    ReDim wvec(nPower2 - 1)
    ReDim xvec(nPower2 - 1)
    ReDim yvec(nPower2 - 1)
    ReDim zerovec(nPower2 - 1)

    iBase = 0
    iSwitch = 1
    delta = 20
    dT = Tyr / nStepT
    erdT = Exp(-r * dT)
    vol2 = vol * vol
    rnmudT = (r - q - 0.5 * vol2) * dT
    SqrtV = vol * Sqrt(Tyr)
    dy = delta * SqrtV / nPower2
    ydisc = Log(K / S)
    yeps = ydisc - vbCeiling(ydisc / dy) * dy
    du = twoPI / (nPower2 * dy)

    For i = 0 To nPower2 - 2 Step 2
        wvec(i) = 1
        wvec(i + 1) = -1
        wtwvec(i) = 1
        wtwvec(i + 1) = -1
    Next i
    wtwvec(0) = 0.5
    wtwvec(nPower2 - 1) = -0.5

    bgvec(0) = -0.5 * nPower2 * dy
    yvec(0) = bgvec(0) + yeps
    uvec(0) = -0.5 * nPower2 * du
    For i = 1 To nPower2 - 1
        bgvec(i) = bgvec(i - 1) + dy
        yvec(i) = bgvec(i) + yeps
        uvec(i) = uvec(i - 1) + du
    Next i

    For i = 0 To nPower2 - 1
        crfi = -0.5 * vol2 * dT * uvec(i) * uvec(i)
        cifi = rnmudT * uvec(i)
        crcfvec(i) = Exp(crfi) * Cos(cifi)
        cicfvec(i) = Exp(crfi) * Sin(cifi)
        evec(i) = Max(K - S * Exp(yvec(i)), 0)
        vvec(i) = evec(i)
        vwtwvec(i) = vvec(i) * wtwvec(i)
        zerovec(i) = 0
    Next i

    For j = nStepT - 1 To 1 Step -1

        a1vec = vbCONVfvec(1, vwtwvec, zerovec, nPower2, iSwitch, iBase)

        For i = 0 To nPower2 - 1
            cravec(i) = a1vec(i, 0) * crcfvec(i) - a1vec(i, 1) * cicfvec(i)
            ciavec(i) = a1vec(i, 0) * cicfvec(i) + a1vec(i, 1) * crcfvec(i)
        Next i

        c1vec = vbCONVfvec(-1, cravec, ciavec, nPower2, iSwitch, iBase)

        For i = 0 To nPower2 - 1
            cvec(i) = erdT * wvec(i) * c1vec(i, 0)
            If cvec(i) > evec(i) Then Exit For
        Next i

        id = i
        If id = 0 Then id = 1

        xdisc = (evec(id - 1) - cvec(id - 1)) * yvec(id) - (evec(id) - cvec(id)) * yvec(id - 1)
        xdisc = xdisc / (evec(id - 1) - evec(id) + cvec(id) - cvec(id - 1))
        xeps = xdisc - vbCeiling(xdisc / dy) * dy
        xShift = xeps - yeps

        For i = 0 To nPower2 - 1
            xvec(i) = bgvec(i) + xeps
            evec(i) = Max(K - S * Exp(xvec(i)), 0)
            crecfi = Cos(xShift * uvec(i))
            ciecfi = Sin(xShift * uvec(i))
            craecfvec(i) = cravec(i) * crecfi - ciavec(i) * ciecfi
            ciaecfvec(i) = cravec(i) * ciecfi + ciavec(i) * crecfi
        Next i

        c1vec = vbCONVfvec(-1, craecfvec, ciaecfvec, nPower2, iSwitch, iBase)

        For i = 0 To nPower2 - 1
            cvec(i) = erdT * wvec(i) * c1vec(i, 0)
            vvec(i) = Max(cvec(i), evec(i))
            vwtwvec(i) = vvec(i) * wtwvec(i)
            yvec(i) = xvec(i)
        Next i

        yeps = xeps

    Next j

    xdisc = 0
    xShift = -yeps
    id = 0.5 * nPower2

    a1vec = vbCONVfvec(1, vwtwvec, zerovec, nPower2, iSwitch, iBase)

    For i = 0 To nPower2 - 1
        crfi = -0.5 * vol2 * dT * uvec(i) * uvec(i)
        cifi = rnmudT * uvec(i)
        crecfi = Exp(crfi) * Cos(cifi + xShift * uvec(i))
        ciecfi = Exp(crfi) * Sin(cifi + xShift * uvec(i))
        craecfvec(i) = a1vec(i, 0) * crecfi - a1vec(i, 1) * ciecfi
        ciaecfvec(i) = a1vec(i, 0) * ciecfi + a1vec(i, 1) * crecfi
    Next i

    c1vec = vbCONVfvec(-1, craecfvec, ciaecfvec, nPower2, iSwitch, iBase)

    vbBermPutCONV = erdT * wvec(id) * c1vec(id, 0)
End Function

Function vbCONVfvec(iFwdInv As Int32, crfvec As Object, cifvec As Object, nPower2 As Int32, iSwitch As Int32, iBase As Int32) As Object
    Dim da As Double, alpha As Double, beta As Double, ar As Double, ai As Double, tr As Double, ti As Double, dar As Double, fi As Double
    Dim blockEnd As Int32, blockSize As Int32, i As Int32, indx As Int32, j As Int32, jk As Int32, n As Int32, nBits As Int32
    Dim fvec(,) As Double
    ReDim fvec(nPower2 - 1, 1)

    nBits = Log(nPower2) / Log2

    For i = 0 To nPower2 - 1
        indx = i
        j = 0
        For jk = 0 To nBits - 1
            j = (j * 2) Or (indx And 1)
            indx = vbIntL(indx, 2)
        Next
        fvec(j, 0) = crfvec(i + iBase)
        fvec(j, 1) = cifvec(i + iBase)
    Next

    blockEnd = 1
    blockSize = 2

    Do While blockSize <= nPower2
        da = iFwdInv * twoPI / blockSize
        alpha = 2 * Sin(0.5 * da) * Sin(0.5 * da)
        beta = Sin(da)

        i = 0
        Do While i < nPower2
            ar = 1
            ai = 0

            j = i
            For n = 0 To blockEnd - 1
                jk = j + blockEnd
                tr = ar * fvec(jk, 0) - ai * fvec(jk, 1)
                ti = ai * fvec(jk, 0) + ar * fvec(jk, 1)
                fvec(jk, 0) = fvec(j, 0) - tr
                fvec(j, 0) = fvec(j, 0) + tr
                fvec(jk, 1) = fvec(j, 1) - ti
                fvec(j, 1) = fvec(j, 1) + ti
                dar = alpha * ar + beta * ai
                ai = ai - (alpha * ai - beta * ar)
                ar = ar - dar
                j = j + 1
            Next

            i = i + blockSize
        Loop

        blockEnd = blockSize
        blockSize = blockSize * 2
    Loop

    If iSwitch = 1 Then
        For i = nPower2 / 2 + 1 To nPower2 - 1
            fi = fvec(i, 0)
            fvec(i, 0) = fvec(nPower2 - i, 0)
            fvec(nPower2 - i, 0) = fi
            fi = fvec(i, 1)
            fvec(i, 1) = fvec(nPower2 - i, 1)
            fvec(nPower2 - i, 1) = fi
        Next i
    End If

    If iFwdInv = -1 Then
        For i = 0 To nPower2 - 1
            fvec(i, 0) = fvec(i, 0) / nPower2
            fvec(i, 1) = fvec(i, 1) / nPower2
        Next i
    End If

    vbCONVfvec = fvec
End Function

Function Max(x1 as Double, x2 As Double) As Double
    If x1 >= x2 Then
        Max = x1
    Else
        Max = x2
    End If
End Function

Function vbCeiling(x As Double) As Double
    If Int(x) = x Then
```

```
        vbCeiling = x
    Else
        vbCeiling = Int(x) + 1
    End If
End Function

Function vbIntL(i1 As Int32, i2 As Int32) As Int32
    vbIntL = Int(i1 / i2)
End Function
```

THE VB.NET FUNCTIONS READY FOR EXCELDNA

```
        <DnaLibrary>
        <![CDATA[

        Imports System.Math

        Public Module MyFunctions

        Const Log2 As Double = 0.693147180559945
        Const twoPI As Double = 6.28318530717959

        Function vbBermPutCONV(S As Double, K As Double, r As Double, q As Double, Tyr As Double, vol As Double, nStepT As Int32,
nPower2 As Int32) As Double
            Dim dT As Double, erdT As Double, vol2 As Double, rnmudT As Double, SqrtV As Double, dy As Double, ydisc As Double, yeps As
Double, du As Double, crfi As Double, cifi As Double
            Dim xdisc As Double, xeps As Double, xShift As Double, crecfi As Double, ciecfi As Double
            Dim delta As Int32, i As Int32, iBase As Int32, id As Int32, iSwitch As Int32, j As Int32
            Dim a1vec As Object
            Dim c1vec As Object

            Dim bgvec() As Double
            Dim ciavec() As Double
            Dim cicfvec() As Double
            Dim cilncfvec() As Double
            Dim ciaecfvec() As Double
            Dim cravec() As Double
            Dim crcfvec() As Double
            Dim crlncfvec() As Double
            Dim craecfvec() As Double
            Dim cvec() As Double
            Dim evec() As Double
            Dim uvec() As Double
            Dim vvec() As Double
            Dim vwtwvec() As Double
            Dim wtwvec() As Double
            Dim wvec() As Double
            Dim xvec() As Double
            Dim yvec() As Double
            Dim zerovec() As Double
            ReDim bgvec(nPower2 - 1)
            ReDim ciavec(nPower2 - 1)
            ReDim cicfvec(nPower2 - 1)
            ReDim cilncfvec(nPower2 - 1)
            ReDim ciaecfvec(nPower2 - 1)
            ReDim cravec(nPower2 - 1)
            ReDim crcfvec(nPower2 - 1)
            ReDim crlncfvec(nPower2 - 1)
            ReDim craecfvec(nPower2 - 1)
            ReDim cvec(nPower2 - 1)
            ReDim evec(nPower2 - 1)
            ReDim uvec(nPower2 - 1)
            ReDim vvec(nPower2 - 1)
            ReDim vwtwvec(nPower2 - 1)
            ReDim wtwvec(nPower2 - 1)
            ReDim wvec(nPower2 - 1)
            ReDim xvec(nPower2 - 1)
            ReDim yvec(nPower2 - 1)
            ReDim zerovec(nPower2 - 1)

            iBase = 0
            iSwitch = 1
            delta = 20
            dT = Tyr / nStepT
            erdT = Exp(-r * dT)
            vol2 = vol * vol
            rnmudT = (r - q - 0.5 * vol2) * dT
            SqrtV = vol * Sqrt(Tyr)
            dy = delta * SqrtV / nPower2
            ydisc = Log(K / S)
            yeps = ydisc - vbCeiling(ydisc / dy) * dy
            du = twoPI / (nPower2 * dy)

            For i = 0 To nPower2 - 2 Step 2
                    wvec(i) = 1
                    wvec(i + 1) = -1
                    wtwvec(i) = 1
                    wtwvec(i + 1) = -1
            Next i
            wtwvec(0) = 0.5
            wtwvec(nPower2 - 1) = -0.5
```

```
bgvec(0) = -0.5 * nPower2 * dy
yvec(0) = bgvec(0) + yeps
uvec(0) = -0.5 * nPower2 * du
For i = 1 To nPower2 - 1
        bgvec(i) = bgvec(i - 1) + dy
        yvec(i) = bgvec(i) + yeps
        uvec(i) = uvec(i - 1) + du
Next i

For i = 0 To nPower2 - 1
        crfi = -0.5 * vol2 * dT * uvec(i) * uvec(i)
        cifi = rnmudT * uvec(i)
        crcfvec(i) = Exp(crfi) * Cos(cifi)
        cicfvec(i) = Exp(crfi) * Sin(cifi)
        evec(i) = Max(K - S * Exp(yvec(i)), 0)
        vvec(i) = evec(i)
        vwtwvec(i) = vvec(i) * wtwvec(i)
        zerovec(i) = 0
Next i

For j = nStepT - 1 To 1 Step -1

        a1vec = vbCONVfvec(1, vwtwvec, zerovec, nPower2, iSwitch, iBase)

        For i = 0 To nPower2 - 1
            cravec(i) = a1vec(i, 0) * crcfvec(i) - a1vec(i, 1) * cicfvec(i)
            ciavec(i) = a1vec(i, 0) * cicfvec(i) + a1vec(i, 1) * crcfvec(i)
        Next i

        c1vec = vbCONVfvec(-1, cravec, ciavec, nPower2, iSwitch, iBase)

        For i = 0 To nPower2 - 1
            cvec(i) = erdT * wvec(i) * c1vec(i, 0)
            If cvec(i) > evec(i) Then Exit For
        Next i

        id = i
        If id = 0 Then id = 1

        xdisc = (evec(id - 1) - cvec(id - 1)) * yvec(id) - (evec(id) - cvec(id)) * yvec(id - 1)
        xdisc = xdisc / (evec(id - 1) - evec(id) + cvec(id) - cvec(id - 1))
        xeps = xdisc - vbCeiling(xdisc / dy) * dy
        xShift = xeps - yeps

        For i = 0 To nPower2 - 1
            xvec(i) = bgvec(i) + xeps
            evec(i) = Max(K - S * Exp(xvec(i)), 0)
            crecfi = Cos(xShift * uvec(i))
            ciecfi = Sin(xShift * uvec(i))
            craecfvec(i) = cravec(i) * crecfi - ciavec(i) * ciecfi
            ciaecfvec(i) = cravec(i) * ciecfi + ciavec(i) * crecfi
        Next i

        c1vec = vbCONVfvec(-1, craecfvec, ciaecfvec, nPower2, iSwitch, iBase)

        For i = 0 To nPower2 - 1
            cvec(i) = erdT * wvec(i) * c1vec(i, 0)
            vvec(i) = Max(cvec(i), evec(i))
            vwtwvec(i) = vvec(i) * wtwvec(i)
            yvec(i) = xvec(i)
        Next i

        yeps = xeps

Next j

xdisc = 0
xShift = -yeps
id = 0.5 * nPower2

a1vec = vbCONVfvec(1, vwtwvec, zerovec, nPower2, iSwitch, iBase)

For i = 0 To nPower2 - 1
        crfi = -0.5 * vol2 * dT * uvec(i) * uvec(i)
        cifi = rnmudT * uvec(i)
        crecfi = Exp(crfi) * Cos(cifi + xShift * uvec(i))
        ciecfi = Exp(crfi) * Sin(cifi + xShift * uvec(i))
        craecfvec(i) = a1vec(i, 0) * crecfi - a1vec(i, 1) * ciecfi
        ciaecfvec(i) = a1vec(i, 0) * ciecfi + a1vec(i, 1) * crecfi
Next i

c1vec = vbCONVfvec(-1, craecfvec, ciaecfvec, nPower2, iSwitch, iBase)
```

```
               vbBermPutCONV = erdT * wvec(id) * c1vec(id, 0)
            End Function

            Function vbCONVfvec(iFwdInv As Int32, crfvec As Object, cifvec As Object, nPower2 As Int32, iSwitch As Int32, iBase As Int32) As
Object
               Dim da As Double, alpha As Double, beta As Double, ar As Double, ai As Double, tr As Double, ti As Double, dar As Double, fi As
Double
               Dim blockEnd As Int32, blockSize As Int32, i As Int32, indx As Int32, j As Int32, jk As Int32, n As Int32, nBits As Int32
               Dim fvec(,) As Double
               ReDim fvec(nPower2 - 1, 1)

               nBits = Log(nPower2) / Log2

               For i = 0 To nPower2 - 1
                        indx = i
                        j = 0
                        For jk = 0 To nBits - 1
                           j = (j * 2) Or (indx And 1)
                           indx = vbIntL(indx, 2)
                        Next
                        fvec(j, 0) = crfvec(i + iBase)
                        fvec(j, 1) = cifvec(i + iBase)
               Next

               blockEnd = 1
               blockSize = 2

               Do While blockSize <= nPower2
                        da = iFwdInv * twoPI / blockSize
                        alpha = 2 * Sin(0.5 * da) * Sin(0.5 * da)
                        beta = Sin(da)

                        i = 0
                        Do While i < nPower2
                           ar = 1
                           ai = 0

                           j = i
                           For n = 0 To blockEnd - 1
                                    jk = j + blockEnd
                                    tr = ar * fvec(jk, 0) - ai * fvec(jk, 1)
                                    ti = ai * fvec(jk, 0) + ar * fvec(jk, 1)
                                    fvec(jk, 0) = fvec(j, 0) - tr
                                    fvec(j, 0) = fvec(j, 0) + tr
                                    fvec(jk, 1) = fvec(j, 1) - ti
                                    fvec(j, 1) = fvec(j, 1) + ti
                                    dar = alpha * ar + beta * ai
                                    ai = ai - (alpha * ai - beta * ar)
                                    ar = ar - dar
                                    j = j + 1
                           Next

                           i = i + blockSize
                        Loop

                        blockEnd = blockSize
                        blockSize = blockSize * 2
               Loop

               If iSwitch = 1 Then
                        For i = nPower2 / 2 + 1 To nPower2 - 1
                           fi = fvec(i, 0)
                           fvec(i, 0) = fvec(nPower2 - i, 0)
                           fvec(nPower2 - i, 0) = fi
                           fi = fvec(i, 1)
                           fvec(i, 1) = fvec(nPower2 - i, 1)
                           fvec(nPower2 - i, 1) = fi
                        Next i
               End If

               If iFwdInv = -1 Then
                        For i = 0 To nPower2 - 1
                           fvec(i, 0) = fvec(i, 0) / nPower2
                           fvec(i, 1) = fvec(i, 1) / nPower2
                        Next i
               End If

               vbCONVfvec = fvec
            End Function

            Function Max(x1 as Double, x2 As Double) As Double
```

```
   If x1 >= x2 Then
            Max = x1
   Else
            Max = x2
   End If
End Function

Function vbCeiling(x As Double) As Double
   If Int(x) = x Then
            vbCeiling = x
   Else
            vbCeiling = Int(x) + 1
   End If
End Function

Function vbIntL(i1 As Int32, i2 As Int32) As Int32
   vbIntL = Int(i1 / i2)
End Function

End Module

]]>
</DnaLibrary>
```

THE VB.NET FUNCTIONS CONVERTED INTO C#

```
private const double Log2 = 0.693147180559945;
private const double twoPI = 6.28318530717959;

public double csBermPutCONV(double S, double K, double r, double q, double Tyr, double vol, Int32 nStepT, Int32 nPower2)
{
    double dT = 0;
    double erdT = 0;
    double vol2 = 0;
    double rnmudT = 0;
    double SqrtV = 0;
    double dy = 0;
    double ydisc = 0;
    double yeps = 0;
    double du = 0;
    double crfi = 0;
    double cifi = 0;
    double xdisc = 0;
    double xeps = 0;
    double xShift = 0;
    double crecfi = 0;
    double ciecfi = 0;
    Int32 delta = 0;
    Int32 i = 0;
    Int32 iBase = 0;
    Int32 id = 0;
    Int32 iSwitch = 0;
    Int32 j = 0;
    object a1vec = null;
    object c1vec = null;

    double[] bgvec = null;
    double[] ciavec = null;
    double[] cicfvec = null;
    double[] cilncfvec = null;
    double[] ciaecfvec = null;
    double[] cravec = null;
    double[] crcfvec = null;
    double[] crlncfvec = null;
    double[] craecfvec = null;
    double[] cvec = null;
    double[] evec = null;
    double[] uvec = null;
    double[] vvec = null;
    double[] vwtwvec = null;
    double[] wtwvec = null;
    double[] wvec = null;
    double[] xvec = null;
    double[] yvec = null;
    double[] zerovec = null;
    bgvec = new double[nPower2];
    ciavec = new double[nPower2];
    cicfvec = new double[nPower2];
    cilncfvec = new double[nPower2];
    ciaecfvec = new double[nPower2];
    cravec = new double[nPower2];
    crcfvec = new double[nPower2];
    crlncfvec = new double[nPower2];
    craecfvec = new double[nPower2];
    cvec = new double[nPower2];
    evec = new double[nPower2];
    uvec = new double[nPower2];
    vvec = new double[nPower2];
    vwtwvec = new double[nPower2];
    wtwvec = new double[nPower2];
    wvec = new double[nPower2];
    xvec = new double[nPower2];
    yvec = new double[nPower2];
    zerovec = new double[nPower2];

    iBase = 0;
    iSwitch = 1;
    delta = 20;
    dT = Tyr / nStepT;
    erdT = Exp(-r * dT);
    vol2 = vol * vol;
    rnmudT = (r - q - 0.5 * vol2) * dT;
    SqrtV = vol * Sqrt(Tyr);
    dy = delta * SqrtV / nPower2;
    ydisc = Log(K / S);
    yeps = ydisc - csCeiling(ydisc / dy) * dy;
```

```
du = twoPI / (nPower2 * dy);

for (i = 0; i <= nPower2 - 2; i = i + 2)
{
    wvec[i] = 1;
    wvec[i + 1] = -1;
    wtwvec[i] = 1;
    wtwvec[i + 1] = -1;
}
wtwvec[0] = 0.5;
wtwvec[nPower2 - 1] = -0.5;

bgvec[0] = -0.5 * nPower2 * dy;
yvec[0] = bgvec[0] + yeps;
uvec[0] = -0.5 * nPower2 * du;
for (i = 1; i < nPower2; i++)
{
    bgvec[i] = bgvec[i - 1] + dy;
    yvec[i] = bgvec[i] + yeps;
    uvec[i] = uvec[i - 1] + du;
}

for (i = 0; i < nPower2; i++)
{
    crfi = -0.5 * vol2 * dT * uvec[i] * uvec[i];
    cifi = rnmudT * uvec[i];
    crcfvec[i] = Exp(crfi) * Cos(cifi);
    cicfvec[i] = Exp(crfi) * Sin(cifi);
    evec[i] = Max(K - S * Exp(yvec[i]), 0);
    vvec[i] = evec[i];
    vwtwvec[i] = vvec[i] * wtwvec[i];
    zerovec[i] = 0;
}

for (j = nStepT - 1; j >= 1; j--)
{

    a1vec = csCONVfvec(1, vwtwvec, zerovec, nPower2, iSwitch, iBase);

    for (i = 0; i < nPower2; i++)
    {
        cravec[i] = a1vec(i, 0) * crcfvec[i] - a1vec(i, 1) * cicfvec[i];
        ciavec[i] = a1vec(i, 0) * cicfvec[i] + a1vec(i, 1) * crcfvec[i];
    }

    c1vec = csCONVfvec(-1, cravec, ciavec, nPower2, iSwitch, iBase);

    for (i = 0; i < nPower2; i++)
    {
        cvec[i] = erdT * wvec[i] * c1vec(i, 0);
        if (cvec[i] > evec[i])
        {
          break;
        }
    }

    id = i;
    if (id == 0)
    {
      id = 1;
    }

    xdisc = (evec[id - 1] - cvec[id - 1]) * yvec[id] - (evec[id] - cvec[id]) * yvec[id - 1];
    xdisc = xdisc / (evec[id - 1] - evec[id] + cvec[id] - cvec[id - 1]);
    xeps = xdisc - csCeiling(xdisc / dy) * dy;
    xShift = xeps - yeps;

    for (i = 0; i < nPower2; i++)
    {
        xvec[i] = bgvec[i] + xeps;
        evec[i] = Max(K - S * Exp(xvec[i]), 0);
        crecfi = Cos(xShift * uvec[i]);
        ciecfi = Sin(xShift * uvec[i]);
        craecfvec[i] = cravec[i] * crecfi - ciavec[i] * ciecfi;
        ciaecfvec[i] = cravec[i] * ciecfi + ciavec[i] * crecfi;
    }

    c1vec = csCONVfvec(-1, craecfvec, ciaecfvec, nPower2, iSwitch, iBase);

    for (i = 0; i < nPower2; i++)
    {
        cvec[i] = erdT * wvec[i] * c1vec(i, 0);
```

```
            vvec[i] = Max(cvec[i], evec[i]);
            vwtwvec[i] = vvec[i] * wtwvec[i];
            yvec[i] = xvec[i];
        }

        yeps = xeps;

    }

    xdisc = 0;
    xShift = -yeps;
    id = 0.5 * nPower2;

    a1vec = csCONVfvec(1, vwtwvec, zerovec, nPower2, iSwitch, iBase);

    for (i = 0; i < nPower2; i++)
    {
        crfi = -0.5 * vol2 * dT * uvec[i] * uvec[i];
        cifi = rnmudT * uvec[i];
        crecfi = Exp(crfi) * Cos(cifi + xShift * uvec[i]);
        ciecfi = Exp(crfi) * Sin(cifi + xShift * uvec[i]);
        craecfvec[i] = a1vec(i, 0) * crecfi - a1vec(i, 1) * ciecfi;
        ciaecfvec[i] = a1vec(i, 0) * ciecfi + a1vec(i, 1) * crecfi;
    }

    c1vec = csCONVfvec(-1, craecfvec, ciaecfvec, nPower2, iSwitch, iBase);

    return erdT * wvec[id] * c1vec(id, 0);
}

public object csCONVfvec(Int32 iFwdInv, object crfvec, object cifvec, Int32 nPower2, Int32 iSwitch, Int32 iBase)
{
    double da = 0;
    double alpha = 0;
    double beta = 0;
    double ar = 0;
    double ai = 0;
    double tr = 0;
    double ti = 0;
    double dar = 0;
    double fi = 0;
    Int32 blockEnd = 0;
    Int32 blockSize = 0;
    Int32 i = 0;
    Int32 indx = 0;
    Int32 j = 0;
    Int32 jk = 0;
    Int32 n = 0;
    Int32 nBits = 0;
    double[,] fvec = null;
    fvec = new double[nPower2,2];

    nBits = Log(nPower2) / Log2;

    for (i = 0; i < nPower2; i++)
    {
        indx = i;
        j = 0;
        for (jk = 0; jk < nBits; jk++)
        {
            j = (j * 2) | (indx & 1);
            indx = csIntL(indx, 2);
        }
        fvec[j, 0] = crfvec(i + iBase);
        fvec[j, 1] = cifvec(i + iBase);
    }

    blockEnd = 1;
    blockSize = 2;

    while (blockSize <= nPower2)
    {
        da = iFwdInv * twoPI / blockSize;
        alpha = 2 * Sin(0.5 * da) * Sin(0.5 * da);
        beta = Sin(da);

        i = 0;
        while (i < nPower2)
        {
            ar = 1;
            ai = 0;
            j = i;
            for (n = 0; n < blockEnd; n++)
            {
                jk = j + blockEnd;
                tr = ar * fvec[jk, 0] - ai * fvec[jk, 1];
                ti = ai * fvec[jk, 0] + ar * fvec[jk, 1];
                fvec[jk, 0] = fvec[j, 0] - tr;
                fvec[j, 0] = fvec[j, 0] + tr;
                fvec[jk, 1] = fvec[j, 1] - ti;
                fvec[j, 1] = fvec[j, 1] + ti;
                dar = alpha * ar + beta * ai;
                ai = ai - (alpha * ai - beta * ar);
                ar = ar - dar;
                j = j + 1;
            }

            i = i + blockSize;
        }

        blockEnd = blockSize;
        blockSize = blockSize * 2;
    }

    if (iSwitch == 1)
    {
        for (i = nPower2 / 2 + 1; i < nPower2; i++)
        {
            fi = fvec[i, 0];
            fvec[i, 0] = fvec[nPower2 - i, 0];
            fvec[nPower2 - i, 0] = fi;
            fi = fvec[i, 1];
            fvec[i, 1] = fvec[nPower2 - i, 1];
            fvec[nPower2 - i, 1] = fi;
        }
    }

    if (iFwdInv == -1)
    {
        for (i = 0; i < nPower2; i++)
        {
            fvec[i, 0] = fvec[i, 0] / nPower2;
            fvec[i, 1] = fvec[i, 1] / nPower2;
        }
    }

    return fvec;
}

public double Max(double x1, double x2)
{
    double tempMax = 0;
    if (x1 >= x2)
    {
        tempMax = x1;
    }
    else
    {
        tempMax = x2;
    }
    return tempMax;
}

public double csCeiling(double x)
{
    double tempcsCeiling = 0;
    if (Microsoft.VisualBasic.Conversion.Int(x) == x)
    {
        tempcsCeiling = x;
    }
    else
    {
        tempcsCeiling = Microsoft.VisualBasic.Conversion.Int(x) + 1;
    }
    return tempcsCeiling;
}

public Int32 csIntL(Int32 i1, Int32 i2)
{
    return Microsoft.VisualBasic.Conversion.Int(i1 / i2);
}
```

THE VB.NET FUNCTIONS CONVERTED INTO C++

```
private:
static const double Log2 = 0.693147180559945;
static const double twoPI = 6.28318530717959;

public:
double cpBermPutCONV(double S, double K, double r, double q, double Tyr, double vol, Int32 nStepT, Int32 nPower2)
{
    double dT = 0;
    double erdT = 0;
    double vol2 = 0;
    double rnmudT = 0;
    double SqrtV = 0;
    double dy = 0;
    double ydisc = 0;
    double yeps = 0;
    double du = 0;
    double crfi = 0;
    double cifi = 0;
    double xdisc = 0;
    double xeps = 0;
    double xShift = 0;
    double crecfi = 0;
    double ciecfi = 0;
    Int32 delta = 0;
    Int32 i = 0;
    Int32 iBase = 0;
    Int32 id = 0;
    Int32 iSwitch = 0;
    Int32 j = 0;
    System::Object ^a1vec = nullptr;
    System::Object ^c1vec = nullptr;

    array<double> ^bgvec = nullptr;
    array<double> ^ciavec = nullptr;
    array<double> ^cicfvec = nullptr;
    array<double> ^cilncfvec = nullptr;
    array<double> ^ciaecfvec = nullptr;
    array<double> ^cravec = nullptr;
    array<double> ^crcfvec = nullptr;
    array<double> ^crlncfvec = nullptr;
    array<double> ^craecfvec = nullptr;
    array<double> ^cvec = nullptr;
    array<double> ^evec = nullptr;
    array<double> ^uvec = nullptr;
    array<double> ^vvec = nullptr;
    array<double> ^vwtwvec = nullptr;
    array<double> ^wtwvec = nullptr;
    array<double> ^wvec = nullptr;
    array<double> ^xvec = nullptr;
    array<double> ^yvec = nullptr;
    array<double> ^zerovec = nullptr;
    bgvec = gcnew array<double>(nPower2);
    ciavec = gcnew array<double>(nPower2);
    cicfvec = gcnew array<double>(nPower2);
    cilncfvec = gcnew array<double>(nPower2);
    ciaecfvec = gcnew array<double>(nPower2);
    cravec = gcnew array<double>(nPower2);
    crcfvec = gcnew array<double>(nPower2);
    crlncfvec = gcnew array<double>(nPower2);
    craecfvec = gcnew array<double>(nPower2);
    cvec = gcnew array<double>(nPower2);
    evec = gcnew array<double>(nPower2);
    uvec = gcnew array<double>(nPower2);
    vvec = gcnew array<double>(nPower2);
    vwtwvec = gcnew array<double>(nPower2);
    wtwvec = gcnew array<double>(nPower2);
    wvec = gcnew array<double>(nPower2);
    xvec = gcnew array<double>(nPower2);
    yvec = gcnew array<double>(nPower2);
    zerovec = gcnew array<double>(nPower2);

    iBase = 0;
    iSwitch = 1;
    delta = 20;
    dT = Tyr / nStepT;
    erdT = Exp(-r * dT);
    vol2 = vol * vol;
    rnmudT = (r - q - 0.5 * vol2) * dT;
    SqrtV = vol * Sqrt(Tyr);
    dy = delta * SqrtV / nPower2;
```

```
ydisc = Log[K / S];
yeps = ydisc - cpCeiling(ydisc / dy) * dy;
du = twoPI / (nPower2 * dy);

for (i = 0; i <= nPower2 - 2; i += 2)
{
    wvec[i] = 1;
    wvec[i + 1] = -1;
    wtwvec[i] = 1;
    wtwvec[i + 1] = -1;
}
wtwvec[0] = 0.5;
wtwvec[nPower2 - 1] = -0.5;

bgvec[0] = -0.5 * nPower2 * dy;
yvec[0] = bgvec[0] + yeps;
uvec[0] = -0.5 * nPower2 * du;
for (i = 1; i < nPower2; i++)
{
    bgvec[i] = bgvec[i - 1] + dy;
    yvec[i] = bgvec[i] + yeps;
    uvec[i] = uvec[i - 1] + du;
}

for (i = 0; i < nPower2; i++)
{
    crfi = -0.5 * vol2 * dT * uvec[i] * uvec[i];
    cifi = rnmudT * uvec[i];
    crcfvec[i] = Exp(crfi) * Cos(cifi);
    cicfvec[i] = Exp(crfi) * Sin(cifi);
    evec[i] = Max(K - S * Exp(yvec[i]), 0);
    vvec[i] = evec[i];
    vwtwvec[i] = vvec[i] * wtwvec[i];
    zerovec[i] = 0;
}

for (j = nStepT - 1; j >= 1; j--)
{

    a1vec = cpCONVfvec(1, vwtwvec, zerovec, nPower2, iSwitch, iBase);

    for (i = 0; i < nPower2; i++)
    {
        cravec[i] = a1vec(i, 0) * crcfvec[i] - a1vec(i, 1) * cicfvec[i];
        ciavec[i] = a1vec(i, 0) * cicfvec[i] + a1vec(i, 1) * crcfvec[i];
    }

    c1vec = cpCONVfvec(-1, cravec, ciavec, nPower2, iSwitch, iBase);

    for (i = 0; i < nPower2; i++)
    {
        cvec[i] = erdT * wvec[i] * c1vec(i, 0);
        if (cvec[i] > evec[i])
          break;
    }

    id = i;
    if (id == 0)
      id = 1;

    xdisc = (evec[id - 1] - cvec[id - 1]) * yvec[id] - (evec[id] - cvec[id]) * yvec[id - 1];
    xdisc = xdisc / (evec[id - 1] - evec[id] + cvec[id] - cvec[id - 1]);
    xeps = xdisc - cpCeiling(xdisc / dy) * dy;
    xShift = xeps - yeps;

    for (i = 0; i < nPower2; i++)
    {
        xvec[i] = bgvec[i] + xeps;
        evec[i] = Max(K - S * Exp(xvec[i]), 0);
        crecfi = Cos(xShift * uvec[i]);
        ciecfi = Sin(xShift * uvec[i]);
        craecfvec[i] = cravec[i] * crecfi - ciavec[i] * ciecfi;
        ciaecfvec[i] = cravec[i] * ciecfi + ciavec[i] * crecfi;
    }

    c1vec = cpCONVfvec(-1, craecfvec, ciaecfvec, nPower2, iSwitch, iBase);

    for (i = 0; i < nPower2; i++)
    {
        cvec[i] = erdT * wvec[i] * c1vec(i, 0);
        vvec[i] = Max(cvec[i], evec[i]);
        vwtwvec[i] = vvec[i] * wtwvec[i];
```

```
            yvec[i] = xvec[i];
        }

        yeps = xeps;

    }

    xdisc = 0;
    xShift = -yeps;
    id = 0.5 * nPower2;

    a1vec = cpCONVfvec(1, vwtwvec, zerovec, nPower2, iSwitch, iBase);

    for (i = 0; i < nPower2; i++)
    {
        crfi = -0.5 * vol2 * dT * uvec[i] * uvec[i];
        cifi = rnmudT * uvec[i];
        crecfi = Exp(crfi) * Cos(cifi + xShift * uvec[i]);
        ciecfi = Exp(crfi) * Sin(cifi + xShift * uvec[i]);
        craecfvec[i] = a1vec(i, 0) * crecfi - a1vec(i, 1) * ciecfi;
        ciaecfvec[i] = a1vec(i, 0) * ciecfi + a1vec(i, 1) * crecfi;
    }

    c1vec = cpCONVfvec(-1, craecfvec, ciaecfvec, nPower2, iSwitch, iBase);

    return erdT * wvec[id] * c1vec(id, 0);
}

System::Object ^cpCONVfvec(Int32 iFwdInv, System::Object ^crfvec, System::Object ^cifvec, Int32 nPower2, Int32 iSwitch, Int32 iBase)
{
    double da = 0;
    double alpha = 0;
    double beta = 0;
    double ar = 0;
    double ai = 0;
    double tr = 0;
    double ti = 0;
    double dar = 0;
    double fi = 0;
    Int32 blockEnd = 0;
    Int32 blockSize = 0;
    Int32 i = 0;
    Int32 indx = 0;
    Int32 j = 0;
    Int32 jk = 0;
    Int32 n = 0;
    Int32 nBits = 0;
    array<double, 2> ^fvec = nullptr;
    fvec = gcnew array<double, 2>(nPower2,2);

    nBits = Log[nPower2] / Log2;

    for (i = 0; i < nPower2; i++)
    {
        indx = i;
        j = 0;
        for (jk = 0; jk < nBits; jk++)
        {
            j = (j * 2) | (indx & 1);
            indx = cpIntL(indx, 2);
        }
        fvec[j, 0] = crfvec(i + iBase);
        fvec[j, 1] = cifvec(i + iBase);
    }

    blockEnd = 1;
    blockSize = 2;

    while (blockSize <= nPower2)
    {
        da = iFwdInv * twoPI / blockSize;
        alpha = 2 * Sin(0.5 * da) * Sin(0.5 * da);
        beta = Sin(da);

        i = 0;
        while (i < nPower2)
        {
            ar = 1;
            ai = 0;

            j = i;
            for (n = 0; n < blockEnd; n++)
            {
                jk = j + blockEnd;
                tr = ar * fvec[jk, 0] - ai * fvec[jk, 1];
                ti = ai * fvec[jk, 0] + ar * fvec[jk, 1];
                fvec[jk, 0] = fvec[j, 0] - tr;
                fvec[j, 0] = fvec[j, 0] + tr;
                fvec[jk, 1] = fvec[j, 1] - ti;
                fvec[j, 1] = fvec[j, 1] + ti;
                dar = alpha * ar + beta * ai;
                ai = ai - (alpha * ai - beta * ar);
                ar = ar - dar;
                j = j + 1;
            }

            i = i + blockSize;
        }

        blockEnd = blockSize;
        blockSize = blockSize * 2;
    }

    if (iSwitch == 1)
    {
        for (i = nPower2 / 2 + 1; i < nPower2; i++)
        {
            fi = fvec[i, 0];
            fvec[i, 0] = fvec[nPower2 - i, 0];
            fvec[nPower2 - i, 0] = fi;
            fi = fvec[i, 1];
            fvec[i, 1] = fvec[nPower2 - i, 1];
            fvec[nPower2 - i, 1] = fi;
        }
    }

    if (iFwdInv == -1)
    {
        for (i = 0; i < nPower2; i++)
        {
            fvec[i, 0] = fvec[i, 0] / nPower2;
            fvec[i, 1] = fvec[i, 1] / nPower2;
        }
    }

    return fvec;
}

double Max(double x1, double x2)
{
    double tempMax = 0;
    if (x1 >= x2)
        tempMax = x1;
    else
        tempMax = x2;
    return tempMax;
}

double cpCeiling(double x)
{
    double tempcpCeiling = 0;
    if (Microsoft::VisualBasic::Conversion::Int(x) == x)
        tempcpCeiling = x;
    else
        tempcpCeiling = Microsoft::VisualBasic::Conversion::Int(x) + 1;
    return tempcpCeiling;
}

Int32 cpIntL(Int32 i1, Int32 i2)
{
    return Microsoft::VisualBasic::Conversion::Int(i1 / i2);
}
```