



Mike Staunton

# Sharpen Up

This month, I will show how VBA code previously converted to VB.NET can be translated into Microsoft's highly favored new C#.NET language

**T**he differences between VB and C# are substantial and worthwhile enough for O'Reilly to publish a pocket reference guide to converting between the two languages. Fortunately most of the differences (lower or mixed case, declaration of vectors and matrices, line termination characters, comments, variable declaration, if statements, loops) can be handled in an automated fashion by commercial software programs such as InstantCSharp. That leaves just a few small wrinkles for us to consider.

The first wrinkle is that C# lacks the exponentiation (^) character. In my VBA code, my need for the ^ character is limited by my preference for alternatives such as Sqrt(x) instead of  $x^{0.5}$  as well as  $x*x$  (for  $x^2$ ) and  $x*x*x$  (for  $x^3$ ). When needed, the ^ character can be replaced by using the Pow function from the System.Math class in the .NET framework. In VBA, one can create a user-defined function that handles exponentiation and hence allow any function calls in VB.NET to work as well.



The second wrinkle also concerns the System.Math class of numeric functions. For instance, the Power function can be called as Math.Pow or just Pow if used in conjunction with the Using System.Math statement at the top of one's program code in VB.NET. But, in C#, the corresponding imports statement cannot refer to mere classes (such as System.Math) but only to namespaces (such as System). Thus we need to use the fuller Math.Pow rather than just Pow if we wish to allow our code to work across both VB and C#.

The third wrinkle is that the Int function that is part of both Excel and VBA carries through to the Visual Basic side of .NET but not to C#. The Int function is a strange hybrid that rounds positive numbers towards zero (as does Floor) but rounds negative numbers away from zero (as does Ceiling). The Floor and Ceiling functions are not part of VBA but are Excel functions and also members of the .NET Math Class. Thus it is not possible to use the Int function across VBA, VB.NET and C#.NET.

## VB. NET CODE

```
Shared Function VBBarrierOutOptionTian#(ByVal iopt%, _
    ByVal idu%, ByVal S#, ByVal K#, ByVal H#, ByVal Rb#, _
    ByVal r#, ByVal q#, ByVal tyr#, ByVal sigma#, _
    ByVal nb%, ByVal ns%)
' Values the discretely-sampled Out Barrier Option
' Based on Tian (1997)
Dim vo#, deltax#, deltaxb#, erdt#, rmqdt#, lam0#, eta0#
Dim lamd#, lnstep#, etad#, pudt#, pmdt#, pddt#, _
    Si#, ibarr#, jbarr#
Dim i%, j%, kb%
Dim vvec() As Object
ReDim vvec(2 * ns)
deltax = tyr / ns
deltaxb = tyr / nb
erdt = Math.Exp(r * deltax)
rmqdt = (r - q - 0.5 * sigma * sigma) * Math.Sqrt(deltax)
lam0 = Math.Sqrt(1.5)
eta0 = Math.Log(S / H) / (lam0 * sigma * Math.Sqrt(deltax))
lamd = eta0 * lam0 / (0.5 + Int(eta0))
lnstep = lamd * sigma * Math.Sqrt(deltax)
etad = Math.Log(S / H) / (lamd * sigma * Math.Sqrt(deltax))
pudt = (1 / (2 * lamd * lamd) + rmqdt / _
    (2 * lamd * sigma)) / erdt
pmdt = (1 - 1 / (lamd * lamd)) / erdt
pddt = (1 / (2 * lamd * lamd) - rmqdt / _
    (2 * lamd * sigma)) / erdt
j = ns
ibarr = j + etad
Si = Math.Log(S) + (ns + 1) * lnstep
For i = 0 To 2 * j
```

```
    Si = Si - lnstep
    If (Math.Sign(ibarr - i) * idu) < 0 Then
        vvec(i) = Rb
    Else
        vvec(i) = Math.Max(iopt * (Math.Exp(Si) - K), 0)
    End If
Next i
kb = nb - 1
For j = ns - 1 To 1 Step -1
    ibarr = j + etad
    jbarr = Math.Abs(j * deltax - kb * deltaxb)
    If jbarr >= deltax Then
        For i = 0 To 2 * j
            vvec(i) = pudt * vvec(i) + pmdt * _
                vvec(i + 1) + pddt * vvec(i + 2)
        Next i
    Else
        For i = 0 To 2 * j
            If (Math.Sign(ibarr - i) * idu) < 0 Then
                vvec(i) = Rb
            Else
                vvec(i) = pudt * vvec(i) + pmdt *
                    vvec(i + 1) + pddt * vvec(i + 2)
            End If
        Next i
        kb = kb - 1
    End If
Next j
Return pudt * vvec(0) + pmdt * vvec(1) + pddt * vvec(2)
End Function
```

### Comments on the converted C# code

In C#, the variables are explicitly initialized with zero values (this is done within the Dim statement in VB). In C#, the object vvec has 2\*ns+1 elements – this is equivalent to defining the upper bound as 2\*ns in the base 0 VB.NET world. VB and VB.NET's Int function translates into the Microsoft.VisualBasic.Conversion.Int function. In C#, temporary variables such as Temp1 are created to replace the use of loop maximum values such as 2\*j that are not fixed. And ... that's it. You're welcome to look amongst other translation programs but the single free program that I found could not handle the numbering of elements in vectors and matrices. InstantCSharp comes as either a free version that will translate limited code snippets, a \$59 version for unlimited code snippets or the full \$159 version that will convert both snippets and

## You're welcome to look amongst other translation programs but the single free program that I found could not handle the numbering of elements in vectors and matrices

whole projects from VB.NET to C#.NET. And the people at InstantCSharp are so helpful - when I first pointed out that the program did not cope properly with type-declaration characters such as # and % it took them only a few hours to correct their code and make available a new version for me to download.

### Getting back to VB.NET

I'm much less concerned with translation from

C# to VB but there is a recent article in msdn magazine by John Robbins. From my brief experience with the free conversion utilities mentioned, one small snag is that they create arrays in VB.NET with one more element than strictly

### REFERENCES

<http://www.tangiblesoftwaresolutions.com/>  
<http://msdn.microsoft.com/msdnmag/issues/04/08/EndBracket/default.aspx>

## CONVERTED C#.NET CODE

```

public static double VBBarrierOutOptionTian(int iopt, int idu,
double S, double K, double H, double Rb, double r, double
q, double tyr, double sigma, int nb, int ns)
{
    // Values the discretely-sampled Out Barrier Option
    // Based on Tian (1997)
    double vo = 0;
    double deltt = 0;
    double delttb = 0;
    double erdt = 0;
    double rmqdt = 0;
    double lam0 = 0;
    double eta0 = 0;
    double lamd = 0;
    double lnstep = 0;
    double etad = 0;
    double pudt = 0;
    double pmdt = 0;
    double pddt = 0;
    double Si = 0;
    double ibarr = 0;
    double jbarr = 0;
    int i = 0;
    int j = 0;
    int kb = 0;
    object[] vvec = null;
    vvec = new object[2*ns + 1];
    deltt = tyr / ns;
    delttb = tyr / nb;
    erdt = Math.Exp(r * deltt);
    rmqdt = (r - q - 0.5 * sigma * sigma) * Math.Sqrt(deltt);
    lam0 = Math.Sqrt(1.5);
    eta0 = Math.Log(S / H) / (lam0 * sigma * Math.Sqrt(deltt));
    lamd = eta0 * lam0 / (0.5 +
        Microsoft.VisualBasic.Conversion.Int(eta0));
    lnstep = lamd * sigma * Math.Sqrt(deltt);
    etad = Math.Log(S / H) / (lamd * sigma * Math.Sqrt(deltt));
    pudt = (1 / (2 * lamd * lamd) + rmqdt /
        (2 * lamd * sigma)) / erdt;
    pmdt = (1 - 1 / (lamd * lamd)) / erdt;
    pddt = (1 / (2 * lamd * lamd) - rmqdt /
        (2 * lamd * sigma)) / erdt;
    j = ns;
    ibarr = j + etad;
    Si = Math.Log(S) + (ns + 1) * lnstep;
    int ForTemp1 = 2 * j;
    for (i = 0; i <= ForTemp1; i++)
    {
        Si = Si - lnstep;
        if ((Math.Sign(ibarr - i) * idu) < 0)
        {
            vvec[i] = Rb;
        }
        else
    }
}

{
    vvec[i] = Math.Max(iopt *
        (Math.Exp(Si) - K), 0);
}
}
kb = nb - 1;
for (j = ns - 1; j >= 1; j--)
{
    ibarr = j + etad;
    jbarr = Math.Abs(j * deltt - kb * delttb);
    if (jbarr >= deltt)
    {
        int ForTemp2 = 2 * j;
        for (i = 0; i <= ForTemp2; i++)
        {
            vvec[i] = pudt *
                vvec[i] + pmdt * vvec[i + 1]
                + pddt * vvec[i + 2];
        }
    }
    else
    {
        int ForTemp3 = 2 * j;
        for (i = 0; i <= ForTemp3; i++)
        {
            if ((Math.Sign(ibarr - i) *
                idu) < 0)
            {
                vvec[i] = Rb;
            }
            else
            {
                vvec[i] = pudt * vvec[i]
                    + pmdt * vvec[i + 1] +
                    pddt * vvec[i + 2];
            }
        }
        kb = kb - 1;
    }
}
return pudt * vvec[0] + pmdt * vvec[1] + pddt * vvec[2];
}

```