Mike Staunton

Convolution for Levy

The CONV method used to value European calls under geometric Brownian motion

Despite the bad publicity that former holder of the yellow jersey Michael Rasmussen brought to cycling and his Rabobank team, I've chosen to write about a paper written by a convolution of Rabobank International and Delft University of Technology. I've long been a fan of numerical integration and I think that this paper combining integration with the fast Fourier transform will still be wearing the yellow jersey (assuming that the Tour de France survives) when the 10th anniversary of *Wilmott* comes around in 2012.

This paper is the culmination of a journey in numerical integration that started with the QUAD paper from the Manchester quartet, developed into QUAD-FFT with O'Sullivan and now the CONV method that can value Bermudan options for a wide range of exponential Levy models (the paper examines VG and CGMY but characteristic functions for other models can easily be substituted). I will illustrate the simplest example where the CONV method is used to value European calls under geometric Brownian motion.

The idea for the CONV paper came from a presentation from Eric Reiner at UCLA in January 2001. Reiner recognized that the standard risk-neutral valuation formula can be seen as a convolution of the continuation value with the transition density. Fourier space is a very convenient place to evaluate convolution integrals since we can use the product

"He's one of those 'Method' modelers ..."

rule for the characteristic function of the sum of two independent random variables. Reiner poses the question that it appears that we've just replaced one quadratic-time algorithm with another and added complex arithmetic to the soup but then introduces the fast Fourier transform to allow a log-linear-time algorithm.

The team from Rabobank and Delft translate Reiner's idea squarely into the Levy world and concentrate on achieving smooth convergence for their CONV algorithm, giving regular second order convergence when combined with trapezium rule integration.

In contrast to their lovely paper, my code looks a bit ugly. The integration points for the trapezium rule are calculated separately for odd and even points within the main loop but there are additional lines of code either side of the loop to cover the first and last points with half weights. After that, there's a single forward FFT to get a1vec (equation 19 in the paper) and a single inverse FFT to get c1vec (equation 18). You'll need to find your own function for the FFT calculations, where mine has been written to match the answers from the fft and ifft functions in Matlab though mine separates the real and imaginary parts of the complex numbers.

The CONV method really comes into its own when valuing Bermudan options but I hope that I've encouraged you to download the paper and follow the team with the yellow jersey.

REFERENCES

Andricopoulos, A.D., Widdicks, M., Duck, P.W., and Newton, D.P., 2003, "Universal Option Valuation Using Quadrature", *Journal of Financial Economics*, 67(3), 447-71.

Lord, Fang F., Bervoets, F., and Oosterlee, C.W., 2007, "A Fast and Accurate FFT-based Method for Pricing Early-Exercise Options under Levy Processes"; http://ssrn.com/abstract=966046.

O'Sullivan, C., 2005, "Path Dependent Option Pricing under Levy Processes"; http://ssrn.com/abstract=673424.

Reiner, E., 2001, "Convolution Methods for Path-Dependent Options"; http://www.ipam.ucla.edu/schedule.aspx?pc=fm2001.

THE VBA CODE

```
Function vaEuroCallCONV(S#, K#, r#, _
       q#, Tyr#, vol#, nPower2L&) _
       As Double
      Dim V#, rnmuT#, dy#, ydisc#, _
        yeps#, xshift#, du#, ui#, _
        yi#, crlncfi#, cilncfi#, vi#
      Dim delta%
      Dim iL&, idL&
      Dim a1vec
      Dim c1vec
      Dim ciecfvec() As Double
      Dim crecfvec() As Double
      Dim cifvec() As Double
      Dim crfvec() As Double
      Dim vwtwvec() As Double
      Dim zerovec() As Double
      ReDim ciecfvec(nPower2L - 1)
      ReDim crecfvec(nPower2L - 1)
      ReDim cifvec(nPower2L - 1)
      ReDim crfvec(nPower2L - 1)
      ReDim vwtwvec(nPower2L - 1)
      ReDim zerovec(nPower2L - 1)

      delta = 20
      idL = 0.5 * nPower2L
      V = vol * vol * Tyr
      rnmuT = (r - q - 0.5 * vol * vol) _
        * Tyr
      dy = delta * Sqr(V) / nPower2L
      ydisc = Log(K / S)
      yeps = ydisc - _
        vaCeiling(ydisc / dy) * dy
      xshift = -yeps
      du = twoPI / (nPower2L * dy)
      ui = -idL * du
      yi = -idL * dy + yeps
      crlncfi = -0.5 * V * ui * ui
      cilncfi = rnmuT * ui
      crecfvec(0) = Exp(crlncfi) * _
        Cos(cilncfi + xshift * ui)
      ciecfvec(0) = Exp(crlncfi) * _
        Sin(cilncfi + xshift * ui)
      vi = Max(S * Exp(yi) - K, 0)
      vwtwvec(0) = 0.5 * vi
      zerovec(0) = 0

      For iL = 1 To nPower2L - 3 Step 2
         ui = ui + du
         yi = yi + dy
         crlncfi = -0.5 * V * ui * ui
         cilncfi = rnmuT * ui
         crecfvec(iL) = Exp(crlncfi) * C_
               os(cilncfi + xshift * ui)
         ciecfvec(iL) = Exp(crlncfi) * _
               Sin(cilncfi + xshift * ui)
         vi = Max(S * Exp(yi) - K, 0)
         vwtwvec(iL) = -vi
         zerovec(iL) = 0
         ui = ui + du
         yi = yi + dy
```

```
         crlncfi = -0.5 * V * ui * ui
         cilncfi = rnmuT * ui
         crecfvec(iL + 1) = Exp(crlncfi) *
_
               Cos(cilncfi + xshift * ui)
         ciecfvec(iL + 1) = Exp(crlncfi) *
_
               Sin(cilncfi + xshift * ui)
         vi = Max(S * Exp(yi) - K, 0)
         vwtwvec(iL + 1) = vi
         zerovec(iL + 1) = 0
      Next iL

      ui = ui + du
      yi = yi + dy
      crlncfi = -0.5 * V * ui * ui
      cilncfi = rnmuT * ui
      crecfvec(nPower2L - 1) = _
        Exp(crlncfi) * _
        Cos(cilncfi + xshift * ui)
      ciecfvec(nPower2L - 1) = _
        Exp(crlncfi) * _
        Sin(cilncfi + xshift * ui)
      vi = Max(S * Exp(yi) - K, 0)
      vwtwvec(nPower2L - 1) = -0.5 * vi
      zerovec(nPower2L - 1) = 0
      a1vec = _
        vaCONVfvec(1, vwtwvec, _
        zerovec, nPower2L)

      For iL = 0 To nPower2L - 1
         crfvec(iL) = a1vec(iL, 0) * _
          crecfvec(iL) - _
          a1vec(iL, 1) * ciecfvec(iL)
         cifvec(iL) = a1vec(iL, 0) * _
          ciecfvec(iL) + _
          a1vec(iL, 1) * crecfvec(iL)
      Next iL

      c1vec = vaCONVfvec(-1, _
        crfvec, cifvec, nPower2L)
      vaEuroCallCONV = Exp(-r * Tyr) * _
        c1vec(idL, 0)
   End Function

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

   Function Max(x, y)
      If x >= y Then
         Max = x
      Else
         Max = y
      End If
   End Function
```