COMPARISON OF VARIANCE RISK PREMIA ACROSS COMMODITIES

By

Maitreyi Mandal

November, 2012

Commodities have emerged as an important asset class in recent years due to many of its attractive properties including a low correlation (stock and bonds) with other major asset classes leading to a great diversifier, hedge against inflation (which many economists view as a real threat in recent times) and also as a currency driver. It is important to gain an understanding of this emerging asset class through a study of risk-return profile. This project starts from the theoretical foundation of variance swap laid in the papers by Carr and Wu (2009) and later applied in Natural Gas and Crude Oil by Trolle and Schwartz (2010); also investigates the similar risk-return profile of a few other commodities to cover the entire commodities spectrum as feasible. This project report investigates the risk-return profile of Crude Oil (CL), Natural Gas (NG), High-Grade Copper (HG), Corn (C) and S&P 500 Equity Index or SPX. These commodities have been selected to represent the broad array of commodities ranging from energy, industrial metal to soft or agricultural. Also, a comparison has been made with SPX to capture the variance swap of stock markets (which is still the most active and liquid market for variance swaps). The time line of this study is 06/01/2007 to 05/30/2012. Variance swaps help investors to get a clean exposure on volatility if they are holding options on commodity futures. And hedge Vega exposure. Also in recent times volatility has emerged as an asset class on its own and many funds have used volatility arbitrage as one of the main trading strategies. The main results of this project can be summarized as follows: i.) the average variance risk premia measured in both dollar terms and in log returns have been found to be negative for a long position with variance swap for all the commodities and SPX. This is not surprising, only indicates that investors dislike states with high variance. ii) Presence of seasonality in variance risk premia measured in dollar terms for Natural Gas, the mean pay-off for variance swap contract initiated during colder months (October to March) is more negative than the mean pay-off of those contracts initiated during warmer months of April to September.

iii) Variance risk-premia in dollar terms is time-varying and correlated with variance swap strike rate (K (t, T)). However, in log return terms, the variance risk premia is more constant for Natural Gas, Corn, Copper and SPX and not highly correlated with variance swap strike rates.
iv) Results from regressing the log excess return to the long side of a variance swap on the annualized log return of the underlying asset over the life of the swap show that for Crude Oil, there exists a negative, statistically insignificant relation, a positive and statistically significant relationship for Natural Gas, High Grade Copper and Corn and finally a negative and statistically significant relationship for SPX.
This report also contains charts of time-series behavior of square root of realized vs. variance swap strike rates for four commodities and SPX along with the pay-off structure in dollar terms (one of the variance risk premia measures considered in this study) for the time period of 2007-2012.

The report contains a total of seven chapters, chapter one contains the introduction to variance swap and commodities as an asset class, literature review is in chapter two, chapter three consists of data and methodology, chapter four describes results, chapter five gives the interpretation of the results found in chapter four, chapter five compares and contrasts between volatility and variance swaps and presents the time series behavior of VIX and VXO, histogram of pay-off of volatility swap (OEX & VXO) Vs. variance swap (SPX & VIX) and chapter seven presents all possible trading strategies that are possible utilizing variance swaps and finally gives a real life example of a fund's (Blackheath Fund LP) return performance utilizing volatility arbitrage as the main trading strategy . There are also three appendices to represent the various option pricing models used (important since they were not covered in any MFE class), SAS & MATLAB codes used and ticker symbology of CME Group & Commodity Research Bureau or CRBTrader.

APPENDIX B: MATLAB AND SAS CODES USED

Matlab Code for Barone-Adesi-Whaley (BAW) Using Newton-Raphson (Used in ATM option) Part A: European Option on commodity

```
function [X] = EurpeanOptionOnCommodity(isCall, S, X, r, b, T, sigma)
%-----------------------------------------------------------------------------
% Calculates price of european option on a commodity
% is Call         = Call = 1, Put = 0
% S               = Price of underlying asset (S=F option on future)
% X               = Strike Price of Option
% r               = Risk free interest rate
% b               = Cost-of-carry rate (b=0 for option on future)
% T               = Time to Maturity
% sigma           = Volatility
%---------------------------------------------------------------------------
d1 = (log(S/X) + (b+0.5*sigma^2)*T) / (sigma * sqrt(T));
d2 = d1 - sigma *sqrt(T);
 % Value of european call option on commodity
if (isCall)
    optionPrice = S * exp((b-r)*T) * normcdf(d1,0,1) - X * exp(-r*T) *
normcdf(d2,0,1);
else
    optionPrice = X * exp(-r*T) * normcdf(-d2,0,1) - S * exp((b-r)*T) *
normcdf(-d1,0,1);
end
X = optionPrice;
```

Part B:

```
function [X] = impliedVolatility_BAW(isCall, marketPrice, S, X, r, b, T)
%---------------------------------------------------------------------------
% Approximates implied volatility for a given option price by inverting
% Barone-Adesi and Whaley (1987) formula
% "Determining the implied volatility [...]", Kutner (1998)
% isCall          = Call = 1, Put = 0
% marketPrice     = Market price of american option
% S               = Price of underlying asset (S=F option on future)
% X               = Strike Price of Option
% r               = Risk free interest rate
% b               = Cost-of-carry rate (b=0 for option on future)
% T               = Time to Maturity
%---------------------------------------------------------------------------
% Estimation of sigma also requires the estimation of S*
% 1) Condition for cirtical commodity price:
%    f(S*,sigma^2) = c(S*,T) + EEP - (S*-X) = 0
% 2) Option market price must equal model price
%    g(S*,sigma^2) = c(S,T) + A2*(S/S*)^q2 - w = 0
%       EEP : Early Exercise Premium
%       w   : Call Option Premium
%       EEP = Early Exercise Premium
% => Find solution with Generalized Newton Method to solve multiple
% nonlinear equations simutaneously:
%  - Start with initial values S*(0) and sigma(0) and
%    iteratively find the solution:
%    | S*(i+1)      | = | S*(i)      | - J_inv * | g(S*(i),sigma^2(i)) |
%    | sigma^2(i+1) |   | sigma^2(i) |           | f(S*(i),sigma^2(i)) |
% start timer
%tic;
     try
         if (isCall)
            % Set initial starting values
            S_Star_0 = S*1.2;
            %sigma_0 = 0.1;
             S_Star = S_Star_0;
             % Makes implementation a lot slower, but seems to produce values
            % for all kinds of options, even deep-in-the-money
            Black_Vol = blkimpv(S, X, r, T, marketPrice, [], [], true);
            Sigma = Black_Vol;
            SigmaSqr = Sigma^2;
            tolerance_f = 1;
            tolerance_g = 1;
            iterationCount = 0;
             while (tolerance_f > 0.00001 && tolerance_g > 0.001 &&
iterationCount < 1000)
                M = (2*r)/(Sigma^2);
                N = (2*b)/(Sigma^2);
                K = 1 - exp(-r*T);
                q2 = 0.5 * ( -(N-1) + sqrt( (N-1)^2 + (4*M)/K ) );
          d1_star = (log(S_Star/X) + (b+0.5*Sigma^2)*T) / (Sigma * sqrt(T));
                d2_star = d1_star - Sigma * sqrt(T);
                N_d1_Star = normcdf(d1_star,0,1);
                %N_d2_Star = normcdf(d2_star,0,1);
                n_d1_Star = normpdf(d1_star,0,1);
                n_d2_Star = normpdf(d2_star,0,1);
                d1 = (log(S/X) + (b+0.5*Sigma^2)*T) / (Sigma * sqrt(T));
                d2 = d1 - Sigma * sqrt(T);
                %N_d1 = normcdf(d1,0,1);
                %N_d2 = normcdf(d2,0,1);
                n_d2 = normpdf(d2,0,1);
                A2 = (S_Star/q2) * ( 1 - exp((b-r)*T) * N_d1_Star );
                 % Calculate f and g
          c_S_Star = EurpeanOptionOnCommodity(1, S_Star, X, r, b, T, Sigma);
          c = EurpeanOptionOnCommodity(1, S, X, r, b, T, Sigma);
   f = c_S_Star + S_Star/q2 * ( 1 - exp((b-r)*T) * N_d1_Star ) - (S_Star - 
X);
                g = c + A2 * ((S/S_Star)^q2) - marketPrice;
                 % Substitution variables
                y1 = 1 - exp((b-r)*T) * N_d1_Star;
                y2 = n_d1_Star / (S_Star * Sigma * sqrt(T));
                y3 = exp((b-r)*T);
                y4 = b/(Sigma^4) - ( ( (N-1)*b + 2*r/K ) / ( Sigma^4 * 
sqrt((N-1)^2 + 4*M/K) ) );
                y5 = X * sqrt(T) * exp(-r*T) * n_d2 / (2*Sigma);
                y6 = -0.5 * ( log(S_Star/X) + b*T ) / ( (Sigma^3 * sqrt(T)) / 
(4*Sigma) );
                y7 = X * sqrt(T) * exp(-r*T) * n_d2_Star / (2*Sigma);
                 % Partial derivatives of 2x2Jacobi matrix
                dg_dS_Star = ((S/S_Star)^q2) * ( (1/q2)*y1 - 
(S_Star/q2)*y2*y3 - q2*A2*(1/S_Star) );
                df_dS_Star = y3 * N_d1_Star + (y1/q2) - (S_Star/q2)*(y2*y3) - 
1;
                dg_dSigmaSqr = y5 + ((S/S_Star)^q2) * ( A2*log(S/S_Star)*y4 - 
(S_Star/q2)*y3*y6*n_d1_Star - y1*y4*(S_Star/(q2^2)) );
                df_dSigmaSqr = y7 - y1*y4*(S_Star/(q2^2)) - 
(S_Star/q2)*y3*y6*n_d1_Star;
                 % Determinant of jacobi matrix
                DET = (dg_dS_Star*df_dSigmaSqr) - (dg_dSigmaSqr*df_dS_Star);
                 % Inverted jacobi matrix
                J_inv = 1/DET * [df_dSigmaSqr -dg_dSigmaSqr; -df_dS_Star 
dg_dS_Star];
                 % Next iteration step
                result = [S_Star;SigmaSqr] - J_inv * [g;f];
                S_Star = result(1);
                SigmaSqr = result(2);
                Sigma = sqrt(SigmaSqr);
               %S_Star = S_Star - 1/DET * (-dg_dSigmaSqr*f + df_dSigmaSqr*g);
                %SigmaSqr = SigmaSqr - 1/DET * (dg_dS_Star*f - df_dS_Star*g);
                % Update tolerance level
                tolerance_f = abs(f);
                tolerance_g = abs(g);
            end
        else
             % Set initial starting values
            S_Star_0 = S*.8;
            %sigma_0 = 0.1;
             % Makes implementation a lot slower, but seems to produce values
            % for all kinds of options, even deep-in-the-money
            Black_Vol = blkimpv(S, X, r, T, marketPrice, [], [], false);
             S_Star = S_Star_0;
            Sigma = Black_Vol;
            SigmaSqr = Sigma^2;
            tolerance_f = 1;
            tolerance_g = 1;
            iterationCount = 0;
             while (tolerance_f > 0.00001 && tolerance_g > 0.01 &&
iterationCount < 1000)
                M = (2*r)/(Sigma^2);
                N = (2*b)/(Sigma^2);
                K = 1 - exp(-r*T);
                q1 = ( -(N-1) - sqrt( (N-1)^2 + (4*M)/K ) ) / 2;
                 d1_star = (log(S_Star/X) + (b+0.5*Sigma^2)*T) / (Sigma *
sqrt(T));
                d2_star = d1_star - Sigma * sqrt(T);
                N_d1_Star = normcdf(-d1_star,0,1);
                %N_d2_Star = normcdf(-d2_star,0,1);
                n_d1_Star = normpdf(-d1_star,0,1);
                n_d2_Star = normpdf(-d2_star,0,1);
                 d1      = (log(S/X) + (b+0.5*Sigma^2)*T) / (Sigma *
sqrt(T));
                d2      = d1 - Sigma * sqrt(T);
                %N_d1 = normcdf(-d1,0,1);
                %N_d2 = normcdf(-d2,0,1);
                n_d2 = normpdf(-d2,0,1);
                A1 = -(S_Star/q1) * ( 1 - exp((b-r)*T) * N_d1_Star );
                 % Calculate f and g
  p_S_Star = EurpeanOptionOnCommodity(0, S_Star, X, r, b, T, Sigma);
                p = EurpeanOptionOnCommodity(0, S, X, r, b, T, Sigma);
 f = p_S_Star - S_Star/q1 * ( 1 - exp((b-r)*T) * N_d1_Star ) - (X - S_Star);
                g = p + A1*((S/S_Star)^q1) - marketPrice;
                 % Substitution variables
                z1 = 1 - exp((b-r)*T) * N_d1_Star;
                z2 = -n_d1_Star * exp((b-r)*T) / (S_Star * Sigma * sqrt(T));
                z3 = ( 0.5*(log(S_Star/X) + b*T)/(Sigma^3*sqrt(T)) ) -
(sqrt(T)/(4*Sigma));
                z4 = ( b + ( ((N-1)*b + 2*r/K) / sqrt((N-1)^2 + 4*M/K) ) ) /
(Sigma^4);
                z5 = (S_Star/q1) * ( exp((b-r)*T) * n_d1_Star * z3 ) +
(S_Star/(q1^2))*z1*z4;
                % Partial derivatives of 2x2Jacobi matrix
                dg_dS_Star = ((S/S_Star)^q1) * ( (S_Star/q1)*z2 - (1/q1)*z1 -
q1*A1*(1/S_Star) );
                df_dS_Star = z1 * (1 - 1/q1) + (S_Star/q1)*z2;
                dg_dSigmaSqr = ( n_d2*X*sqrt(T)*exp(-r*T) / (2*Sigma) ) +
((S/S_Star)^q1) * ( A1*log(S/S_Star)*z4+z5 );
                df_dSigmaSqr = ( n_d2_Star*X*sqrt(T)*exp(-r*T) / (2*Sigma) )
+ (S_Star/q1)*exp((b-r)*T)*n_d1_Star*z3 + (S_Star/(q1^2))*z1*z4;


                % Determinant of jacobi matrix
                J = [dg_dS_Star dg_dSigmaSqr; df_dS_Star df_dSigmaSqr];
                DET = det(J);
                 % Inverted jacobi matrix
                J_inv = 1/DET * [df_dSigmaSqr -dg_dSigmaSqr; -df_dS_Star
dg_dS_Star];
                % Next iteration step
                stepsize = .5;
                result = [S_Star;SigmaSqr] - stepsize * J_inv * [g;f];
                S_Star = result(1);
                SigmaSqr = result(2);
                Sigma = sqrt(SigmaSqr);
                % Update tolerance level
                tolerance_f = abs(f)/X;
                tolerance_g = abs(g);
            end
        end
    catch
        Sigma = NaN;
        S_Star = NaN;
    end
     % end timer
    %toc;
     %X = [Sigma;S_Star];
     % Return implied volatiliry
    X = Sigma;
 end
```

SIMPSON's Rule of INTEGRATION

```
function I = simpsons(f,a,b,n)
% This function computes the integral "I" via Simpson's rule in the interval
[a,b] with n+1 equally spaced points
%
% Syntax: I = simpsons(f,a,b,n)
%
% Where,
%  f= can be either an anonymous function (e.g. f=@(x) sin(x)) or a vector
%  containing equally spaced values of the function to be integrated
%  a= Initial point of interval
%  b= Last point of interval
%  n= # of sub-intervals (panels), must be integer
if numel(f)>1 % If the input provided is a vector
    n=numel(f)-1; h=(b-a)/n;
    I= h/3*(f(1)+2*sum(f(3:2:end-2))+4*sum(f(2:2:end))+f(end));
else % If the input provided is an anonymous function
    h=(b-a)/n; xi=a:h:b;
    I= h/3*(f(xi(1))+2*sum(f(xi(3:2:end-
2)))+4*sum(f(xi(2:2:end)))+f(xi(end)));
end
```

Part B: SAS Codes Used

Generalized Method of Moment (GMM)

```
proc model data=File1;
         endo VCL;
         exog KCL;
         */instruments _exog_;
         parms b0 b1 ;
         VCL=b0 + b1*KCL;
         fit VCL / gmm kernel=(bart,5,0) vardef=n;
         run;
         quit;
```

EGARCH (1,1) Estimation

```
proc autoreg data= File1;
 model RSPX=   / garch= ( q=1, p=1 , type = exp) ;
                    output out=a predicted=p residual=r cev=v ;
 run;
```

Finding out Correlation

```
proc corr data=Sasuser.corr nomiss outp=CorrOutp;
  var DKNG DKHG DKSPX DVNG DVHG DVSPX RCop RNG RSPX;
run;
```