The Journal of Undergraduate Research

Volume 10 Article 13

2013

# Stock Index Options Pricing Models

Mark York

# Stock Index Options Pricing Models

Author: Mark York

Faculty Sponsors: Dr. Zhiguang Wang and Dr. Jung-Han Kimn

Departments: Mathematics & Statistics and Economics

# ABSTRACT

The purpose of this research is to apply stochastic modeling methods to determine the prices of stock index options. In this paper, three models are implemented and compared for accuracy based on the S&P 500 index (SPX) options data for 1996. These models include the Black-Scholes Model (BS), a stochastic volatility model (SV) which accounts for volatility in the underlying stock price, and a stochastic volatility model with jump in the underlying stock price (SVJ). This jump in the stock index prices is accounted for in the SVJ model using a compound Poisson distribution. The SV model is nested in the SVJ model, with jump-related parameters being set to zero. These three mathematical models are implemented in MATLAB. The models are calibrated using data from one day at a time for the SPX options on the Chicago Board Options Exchange based on the criteria of the least sum of squared errors. The models are tested for consistency by calibrating and measuring the squared error for each day in an entire year's worth of option price data.

**Keywords:** Computational Finance, Stock, Options, SPX Index, Black-Scholes

# INTRODUCTION

According to the Chicago Board Options Exchange (CBOE), S&P 500 index (SPX) options have been and still are the most actively traded options contracts in the world. SPX options are employed by both financial institutions and individual investors for the purposes of risk management and speculations. In this paper, three models for pricing SPX options are compared empirically in order to decide which model is most appropriate for index options pricing.

The reason why SPX options are popular among traders is largely due to the popularity of the underlying index, the S&P 500 index, which is the media-favorite for the overall stock market activity in the US. Stock index options, like individual stock options can provide holders with a right, not an obligation to take long (for call options) or short (for put options) positions on the underlying financial asset. Holding put options can provide downside protection for investors, whereas holding call options can allow investors to take advantage of the upside potential in the future. Therefore, options can be employed to express views about the stock market without immediate possession of the underlying stock index (or 500 stocks in the case of the SPX).

The most important question for both buyers and sellers of option contracts is how much these options should be bought and sold for? Both call and put options are valuable before they expire on the maturity date, usually the third Friday of each calendar month. The value of an option on the maturity date is max(Asset Price-Exercise Price, 0)for a call and is max(Exercise Price-Asset Price, 0) for a put. Before the maturity date, the option price fluctuates over time as the underlying stock prices are dynamically set by the market force.

In 1973, Fischer Black and Myron Scholes developed a model later known as the Black-Scholes model (BS) to determine what the price of options should be. With this revolutionary new model, investors had a concrete mathematical way of determining what price they should buy and sell their options for, rather than relying on intuition. However, over the years, it became clear that the model generated inaccuracies from time to time. There are two major deviations of the assumptions of the BS model: the variability of asset returns is constant and there is no discontinuity in asset prices. To account for these shortcomings, stochastic volatility and jumps are added to the original BS model (see [1] Bakshi, Cao and Chen 1997 and [2] Duffie, Pan and Singleton 2000), which are called SV and SVJ models, respectively.

The purpose of this research is to show how the BS model, along with the SV and SVJ models, can be implemented using computer programs to model a real market situation, and to compare these models in order to find a model that best fits the data. Option traders can employ the methodology derived for the best fitting model in their trading practices.

# METHODS

The BS, SV, and SVJ models for index options are described as follows. The log of the underlying stock index price is denoted as *y*, the strike price *c*, the time to maturity *T* (in years), and options price $X_0$.

$y$ = *natural log of the stock price*

$c$ = *strike price*

$T$ = *time to maturity* (*in years*)

$X_0$ = *option price*

For simplicity, options data of daily frequency is employed. Therefore the end-of-day closing prices of the stock index and its options are the relevant prices. Several hundred options contracts with various combinations of strike price and time to maturity are listed on any given day.

The BS model has only one input, $\sigma$, also called implied volatility. The infamous Black-Scholes formula for call option price is

$$C(S,t) = N(d_1)S - N(d_2)ce^{-r(t-T)}, \tag{1}$$

where $N(d_1)$ represents the cumulative standard normal distribution of $d_1$, $c$ represents the strike price, $r$ is the annual risk-free interest rate, which is assumed constant at 0.0319, ([1] Bakshi, Cao, and Chen 1997), $t$ represents the current time, which is set at 0, and S represents the stock price (or equivalently $e^y$). Within the cumulative normal distributions, $d_1$ and $d_2$ are defined as

$$d_1 = \frac{ln(\frac{S}{c}) + (r + \frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{T-t}}$$

$$d_2 = \frac{ln(\frac{S}{c}) + (r - \frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{T-t}} = d_1 - \sigma\sqrt{T-t}. \tag{2}$$

The price of a put option can be derived from the call-put parity equation, which is written as

$$P(S,t) = Ke^{-r(T-t)} - S + C(S,t). \tag{3}$$

In (3) above, S and $C(S,t)$ denote the stock price and the call option price calculated from Equation (1), respectively.

The SV and SVJ models consist of the variables listed above (*y, c, T* and $X_0$), along with eight other parameters that characterize the features of stochastic volatility and jumps. These parameters in the SVJ model and the SV model (in parenthesis) are listed below:

$\bar{\rho} = Leverage\ effect$

$\bar{v} = Long\ run\ mean\ volatility$

$\kappa_v = Speed\ of\ mean\ reversion$

$\sigma_v = Volatility\ of\ volatility$

$\sigma_y = Standard\ deviation\ of\ stock\ price\ jump\ (set\ to\ 0\ in\ SV)$

$\lambda_y = Poisson\ rate\ parameter\ for\ stock\ price\ jumps\ (0\ in\ SV)$

$\mu_y = Mean\ jump\ size\ (0\ in\ SV)$

$V = Volatility\ Squared$

The pricing function for call options for the SV and SVJ models follows[1] Bakshi, Cao and Chen (1997) and [2] Duffie, Pan and Singleton (2000) appears as

$$C(d,c,T,y,V) = G_{d,-d}(-\ln(c), X_0, T) - cG_{0,-d}(-\ln(c), X_0, T), \tag{4}$$

where *d* is set to 1 for call options. The variables *c*, *y*, *T*, and $X_0$ are inputs obtained from the raw data as stated above. The parameter *V* represents squared volatility and is set to match the data during the calibration process. *G* is defined as follows:

$$G_{a,b}(h, X_0, T) = \frac{\Psi(a, X_0, 0, T)}{2} - \frac{1}{\pi}\int_0^{\infty} \frac{Im\left[\Psi(a + ivb, X_0, 0, T)e^{-ivh}\right]}{v} dv, \tag{5}$$

where *Im* inside the integral is the imaginary part of the function in brackets, and Ψ is a moment-generating function of the random variable $X_0$,given by:

$$\Psi(u, X_0, t, T) = e^{(\alpha(T-t,u)+uy+\beta(T-t,u)\cdot V)}. \tag{6}$$

The integral in (5) is difficult to compute, but may be approximated a number of different ways. The approximation used in this project is the Gauss-Laguerre Quadrature, which is described in the next section. Note that the *u* in (6) coordinates with the same position in the G equation, *t* denotes the current time, and *T* is the expiration time. Parameters $\alpha(.)$ and $\beta(.)$ in Equation (6) are solutions to two ordinary differential equations (ODE) detailed in [2] Duffie, Pan, and Singleton (2000). The results read as follows:

$$\alpha(\tau, u) = \alpha_0(\tau, u) - \bar{\lambda}\tau(1 + \bar{\mu}u) + \bar{\lambda}\int_0^{\tau}\theta(u, \beta(s, u))ds,$$

$$\beta(\tau) = \frac{a(1 - e^{-\gamma\tau})}{2\gamma - (\gamma + b)(1 - e^{-\gamma\tau})} \tag{7}$$

by letting:

$$b = \sigma_v\bar{\rho}u - k_v,\ a = u(1-u),\ \gamma = \sqrt{b^2 + a\sigma_v^2},\ \tau = T - t,$$

$$\bar{\mu} = \theta(1,0) - 1 = exp(\mu_y + \frac{1}{2}\sigma_y^2) - 1.$$

$$\alpha_0(\tau, u) = -r\tau + (r - \bar{\zeta})u\tau - \kappa_v\bar{v}(\frac{\gamma + b}{\sigma_v^2}\tau + \frac{2}{\sigma_v^2}ln[1 - \frac{\gamma + b}{2\gamma}(1 - e^{-\gamma\tau})]) \tag{8}$$

In Equations (7-8), *r* is the annual risk-free interest rate and $\bar{\zeta}$ is the dividend yield, which is assumed to be zero ([1] Bakshi, Cao and Chen 1997).

The pricing formula for the SVJ model differs from its counterpart for the SV model in the jump specification. Two parameters are necessary to model a jump: intensity and size. The former determines whether a jump occurs whereas the latter controls the magnitude of the jump. The Poisson jump intensity $\bar{\lambda}$ in the SVJ model is aggregated from three parts, the jump intensity of the stock price $(\lambda_y)$ plus the jump intensity of volatility $(\lambda_v)$ plus the coordinated jump intensity parameter $(\lambda_c)$. The coordinated jump intensity parameter accounts for the fact that a sudden drop in stock price often leads to or is evidence of an increase in market volatility. Since all three are Poisson random variables, the combined rate at which any of them occur is just the sum of the rate parameters. Jumps are presumed to occur whenever the aggregated jump intensity $\bar{\lambda}$ attains a positive value, and the magnitude of the jump is determined by a particular distribution for each of the three types of jump. For simplicity, this model only incorporates jumps in *y* the log of the stock price. For $\lambda_y$, the magnitude of the jump is normally distributed with mean $\mu_y$ and variance $\sigma_y^2$, and both parameters are adjusted to model the stock price data. Therefore, $\lambda_v = \lambda_c = 0$ and $\bar{\lambda} = \lambda_y$, the integral part of α in Equation (7) is defined below.

$$\int_0^\tau \theta(u,\beta(s,u))ds = \bar{\lambda}^{-1}(\lambda_y f_y(u,\tau) + \lambda_v f_v(u,\tau) + \lambda_c f_c(u,\tau))$$

$$= \frac{1}{\lambda_y}(\lambda_y f_y(u,\tau)) = f_y(u,\tau) \quad , \tag{9}$$

where $f_y(u,\tau)$ is defined as

$$f_y(u,\tau) = \tau * exp(\mu_y u + \frac{1}{2}\sigma_y^2 u^2) \quad . \tag{10}$$

The stochastic volatility model with jumps in *y* is formed by calibrating the eight parameters shown in above. The SV model without jumps can be formed by fixing $\lambda_y$ and $\mu_y$ at a value of zero. The original Black-Scholes model from 1973 cannot be formed by simply fixing certain parameters at a value of zero, as this causes DIV/O (division by zero)

situations, and thus it has to be implemented separately using Equation (1). In the next section, the programming and calibration of all three models are discussed.

# IMPLEMENTATION

The model is calibrated to fit the SPX stock options data using the criterion of minimum SSE (Sum of Squared Errors). The eight parameters are calibrated separately for each day by using the data from that day, starting with the first trading day of 1996 (January 4th). The accuracy of the model is measured by finding the average SSE per option for each model. The first trading day has 279 observations, including 139 calls and 140 puts. Calls and puts are modeled separately, as their prices change in the opposite direction as the strike price changes. Using the equations above, the SSE criterion seeks to minimize

$$SSE = \Sigma_{k=1}^{N}(C(d,c(k),T(k),y(k),v) - X_0(k))^2. \quad (11)$$

where $C(d,c(k),T(k),y(k),v)$ is the model's predicted option price for the k$^{th}$ option, and $X_0(k)$ is the actual price for the k$^{th}$ option.

The SSE is calculated by programming the compound function described in the previous section into MATLAB code. In MATLAB, a main function and a series of five sub-functions are programmed which call one another. Figures 1 and 2 describe how the functions are called and how the data is passed from one function to another (see the arrow) for the BS model and the SV/SVJ model, respectively. A detailed explanation of functions involved in the calibration procedure is reported in Table 1.

The program for the BS model starts with the BlackScholes function, which retrieves data from BSReadData, then sets an initial value for sigma. BlackScholes then passes the data for the first day in vector form, along with sigma, to Square, which retrieves the model prices for each option from BSCalculations, calculates the SSE for the day, and passes it to BlackScholes. BlackScholes then tries another value for sigma, and repeats itself until it finds a convergent minimum for the SSE, at which point it moves to the next day.

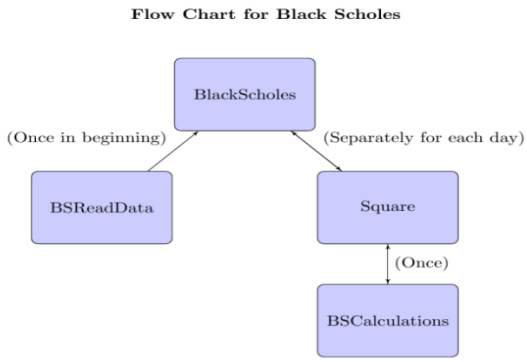


**Figure 1: Flow Chart for the BS Model**

Similarly, for the SV and SVJ models in Figure 2, the program starts with the Main function, which calls retrieves data from ReadData, then sets initial parameter values. Main then passes one day's vectorized data to Sq, which passes the data to Cprice. Cprice passes the data to PsiFunc, which solves most of the above equations and passes their values to CPrice, which then passes the model price to Sq, which calculates the SSE and passes it to Main. Main then tries to set better parameter values to reduce the SSE, and once again calls Sq, proceeding in this way until it finds a convergent minimum for each day.

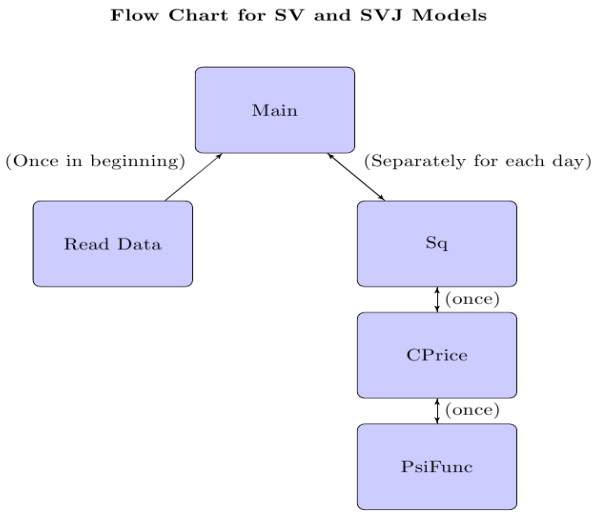


**Figure2: Flow Chart for the SV and SVJ Models**

**Function Description Table**

| Function Name | Description |
|---|---|
| **Black Scholes** | |
| BlackScholes | Calls BSReadData, modifies sigma to find min SSE for each day |
| BSReadData | Retrieves stock and option data from .txt files |
| Square | Finds the SSE for a given day and value of sigma |
| BSCalculations | Calculates the model option prices for one day at a time |
| **SV and SVJ** | |
| Main | Calls Read Data, modifies parameters to find min SSE for each day |
| Read Data | Retrieves stock and option data from .txt files |
| Sq | Finds the SSE for a given day and set of parameters |
| CPrice | Calculates model prices for one day's options and passes them to Sq |
| PsiFunc | Runs function $\psi$ and all sub functions, passes data to CPrice |

**Table 1: Description of MATLAB Functions for the Calibration Procedure**

**Algorithm 1** Pseudocode for Stock Options Pricing Models

**Require:** $[y, X_0, c, T, Call/Put]$

$N$ = number of trading days in year
**for** $1 \leq k \leq N$ **do**
  $q$ = number of calls in day $k$; $p$ = number of puts in day $k$
  **for** $1 \leq m \leq 2$ **do**
    **while** $SSE(m)$ does not satisfy $fmincon$ minimization criteria **do**
      $fmincon$ sets new parameter values $\Phi()$
      $PsiFunc() = (PsiFunc(\Phi()))$
      $ModelPrice() = CPrice(CPrice()); n = q$
      **if** m = 2 **then**
        $ModelPrice() = PPrice(PsiFunc()); n = p$
      **end if**
      **for** $1 \leq h \leq n$ **do**
        $SSE(m) = SSE(m) + (ModelPrice(h) - X_0(h))^2$
      **end for**
    **end while**
  **end for**
  $CallsErr = CallsErr + SSE(1); SSE(1) = 0$
  $PutsErr = PutsErr + SSE(2); SSE(2) = 0$
**end for**
**OUTPUT** $CallsErr, PutsErr$

**Figure 3: Pseudocode for Options Pricing Models**

The pseudocode in Figure 3 shows how the functions in the SV and SVJ program call each other and produce the result. The most critical part of the pricing procedure is to evaluate the function G in Equation (5). As mentioned in the Methods section, the Gauss-Laguerre Quadrature is used to approximate the following integral in the function G.

$$\int_0^{\infty} \frac{Im\left[\Psi(a+ivb, X_0, 0, T)e^{-ivh}\right]}{v} dv$$

The above integral is difficult to solve analytically as it is a compound function composed of large sub-functions. One alternative is to use the Monte Carlo simulation, which generates thousands of random values of *v* and integrates them over a permissible range.

The inaccuracy due to the randomness and truncation of the domain of integration approaches zero as the number of iterations approaches infinity. Unfortunately, this method requires running thousands of iterations of code and slows the process down to the degree that the function would be all but unusable. As such, it was decided to seek another algorithm.

The method decided upon is the Gauss-Laguerre Quadrature ([3] Kythe and Schäferkotter 2005). The Gauss-Laguerre Quadrature approximates the integral of a function by choosing and integer value $n \geq 2$ and $n$ coordinating values of $x_i$ where the function is evaluated. These values of $x_i$ are known as nodes. The function value at each of these nodes is then multiplied by $w_i$, a weight function. The weighted function evaluations at the different nodes are then added up to give the approximation of the integral. As $n$ increases, the error of the approximation approaches zero for most functions. The Gaussian Quadrature is highly regarded as being very accurate, widely applicable, quick, and easy to use. The formula appears below.

$$\int_a^b f(x)dx \approx \Sigma_{i=1}^{n} w_i f(x_i)$$

The problem with the Gaussian Quadrature is that *a* and *b* must be finite, while the improper integral that needs to be evaluated is over the interval $[0,\infty)$. Thus, a modification to the above formula must be made. This modified Gaussian Quadrature is known as the Gauss-Laguerre Quadrature, or Laguerre Integration. The Gauss-Laguerre Quadrature is expressed as

$$\int_0^\infty f(x)dx = \int_0^\infty e^{-x}[e^x f(x)]dx \approx \Sigma_{k=1}^{n} w(x_k)e^x f(x_k).$$

The function $e^{-x}$ will approach zero as *x* approaches, so without delving into too much detail, the $e^{-x}$ term allows us to apply the Gaussian Quadrature to the entire interval, similar to a Laplace Transform. The weights ($w_i$) and nodes ($x_i$) can be determined by formulas defined by Edmond Laguerre which depend solely on *n*. For this paper, a value of 15 was used, as each additional node requires another two values to be run through PsiFunc each time an option is priced, which occurs millions of times during the program. Using a value

of 15 allows for sufficient accuracy while not compromising too much speed. The method whereby the parameters are changed to minimize the sum in (17) involved using two minimization functions in MATLAB. These functions are fminsearch and fmincon. The initial values for the eight parameters are taken from [2] Duffie, Pan, and Singleton (2000). The function fminsearch uses the Nelder-Mead Simplex method, which in general terms develops a simplex or set of n+1 parameter values and evaluates the function at each of these values. It then systematically makes a new set of parameters based on the reflection of the minimizing set about the center of gravity of the rest of the points except the worst point. Following a set of algorithms, it eventually converges to a point where the parameters and function values are minimized, at least locally. Fminsearch is highly effective at finding local minima, but will regularly overlook a global minimum by finding a local minimum.

The function fmincon uses four different algorithms; the algorithm which is used in this program is the Active-Set Algorithm, due to its ability to take large steps, and the fact that it can handle both bounds (used to keep the function value reasonable and focus the optimization), and the equality constraint . This function is an iterative method and calculates the Hessian and Lagrangian for each set of parameters values that is used. The Karush-Kuhn-Tucker equations are involved in calculating the gradients, which are used to find potentially better sets of parameters, which are then used and the method is repeated until a satisfactory minimum is obtained. This method works very well at finding the minimum within a set of bounds. Thus, to increase the chance of finding the global minimum, the bounds must be wide.

However, having a really wide set of bounds causes the program to generate numbers that are out of range based on the unrealistic combinations of parameter values it can generate. For example, it may take the maximum possible value for the first three parameters, then the minimum for the last five, which would create an SSE of 10^300, a number which is out of range and causes the program to crash. As such, a narrower range of bounds must be used, meaning that the initial guess must be close to the actual solution. To check a wide range of possible combinations of bounds for the eight parameters, literally hundreds of different sets of bounds are tried using both fminsearch and fmincon, and using the information from these functions to pick new parameters. Once a set of values that

generates a reasonably small SSE is found, fmincon is used to find the local minimum, which fminsearch and hours of experimentation has shown to be a likely, but not a guaranteed, global minimum.

# RESULTS

In this project, three different models are employed to fit two sets of options data (38,321 calls and 38,156 puts) to generate essentially six sets of results. All three models are very stable, converging to the same answer with a wide range of initial values for calls and puts. This leads one to believe that both the model and the calibration are robust. The summary of these pricing errors is detailed in Table 2. To put the pricing errors in perspective, the mean stock price over the period is $670.89 per share, the mean call option price is $69.09, and the mean put option price is $18.07 per share. On average, the SV model generates the error of 2.29% for call options whereas the SVJ model produces the error of 2.11%. Figure 4 further shows the distribution of squared errors for calls as modeled by the SV model for each of the 252 days. The calls are divided into 10 groups determined by ranges defined as 1/10 of the distance between the maximum and minimum error. While this is just the distribution for one of the six models, it is representative of the distribution in the other cases.

The mean values of sigma for the BS model for calls and puts are 0.151 and 0.146, respectively, for the 252 days for which they are optimized. These values are consistent with the actual observation of the market during the sample period. This indicates the BS model can generate a ballpark, though inaccurate, estimate consistent with the actual market behavior. The parameter estimates for the SV and SVJ models are reported in Table 3. The first four parameters in Table 3 are for stochastic volatility and are statistically significant. The next two parameters indicate that jumps are rare and negative on average, meaning downward jumps outweighing upward jumps. The last row is the implied volatility parameter, the results for which show consistency among all four combinations of models (SV and SVJ, Calls and Puts). To sum up, all parameter estimates are consistent with our initial assumption: the presence of stochastic volatility and jumps.

**Measure of Accuracy**

| Calls | BS | SV | SVJ |
|---|---|---|---|
| **SSE** | 190050 | 60271 | 55732 |
| **SSE/option** | 4.97 | 1.58 | 1.46 |
| **Puts** | | | |
| **SSE** | 318098 | 28873 | 15338 |
| **SSE/option** | 8.23 | 0.75 | 0.4 |

**Table 2: Pricing Errors for the BS, SV and SVJ Models**

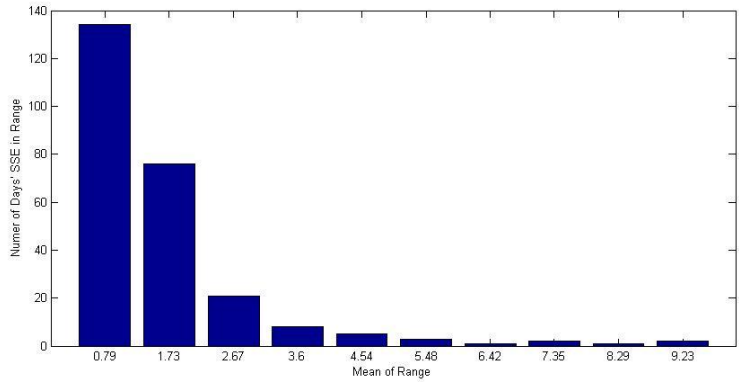


**Figure 4. SV Model Calls – SSE Distribution**

**Mean Parameter Values - 252 Days**

| Parameter | SV Calls | SVJ Calls | SV Puts | SVJ Puts |
|---|---|---|---|---|
| $\overline{\rho}$ | -0.858 | -0.825 | -0.893 | -0.796 |
| $\overline{v}$ | 0.029 | 0.021 | 0.023 | 0.01 |
| $\kappa_v$ | 3.785 | 4.093 | 1.633 | 1.931 |
| $\sigma_v$ | 0.3 | 0.214 | 0.272 | 0.17 |
| $\sigma_y$ | 0 | 0.001 | 0 | 0.112 |
| $\lambda_y$ | 0 | 0.081 | 0 | 0.08 |
| $\mu_y$ | 0 | -0.179 | 0 | -0.429 |
| $\sqrt{V}$ | 0.152 | 0.146 | 0.154 | 0.143 |

**Table 3: Parameter Estimates for the SV and SVJ Models**

Although Table 2 provides preliminary evidence for the relatively better performance of the SVJ model vs. the SV and BS models on average, the details are still yet to be revealed about their relative day-to-day performance. Figure 5 plots a daily comparison of the SSE for each of the three models. It is clear that the BS model generates highest errors while the SVJ and SV models have significantly lower errors. The SVJ model slightly outperforms the SV model.

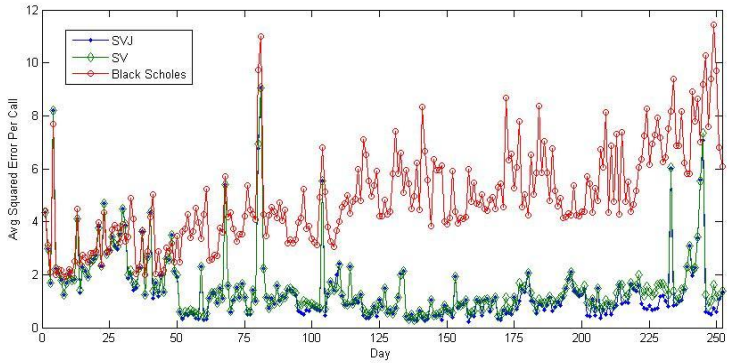


**Figure 5.Daily Comparison of Squared Error**

To determine the level of certainty of the SSE results, a hypothesis test is conducted using the Diebold-Mariano (DM) statistic. The null hypothesis was that the mean squared error for the SV model is the same as that for the SVJ model. For calls, the DM statistic was 30.33, meaning that at the 99.9% confidence level, the null hypothesis is rejected. For puts, the DM statistic is 61.86, leading to a rejection of the null hypothesis at the 99.9% level. This agrees with the t-test, which is also run on the SSE day by day, and generates a t-statistic of 7.1, leading to rejection of the null hypothesis at the 99.9% level. For the variance calculated by the SV and SVJ models when modeling put options, a t-test is run and generates a t-statistic of 5.98, leading to a rejection of the null hypothesis at the 99.9% level. The results show that the errors produced by the SVJ model are lower than those produced by the SV model at the confidence level of 99.9%, although the difference between the two models is small.

# CONCLUSIONS

Based on the analysis of the empirical results, the SVJ model has a significantly greater level of accuracy than the SV model, and the SV model is significantly better than the BS model (although the DM statistics are not presented above, they are greater than those between SV and SVJ). These results held true when modeling both calls and puts. As SPX is the most actively traded European option on the index of 500 common stocks, the modeling technique can be applied to most other European style financial (stock, bonds and foreign exchange) options.

# LIMITATIONS

Areas that could be further studied to improve the model include programming in C code rather than MATLAB, using the MCMC minimization rather than fminsearch and fmincon, and using multiple years and stocks to further validate the results.

MATLAB code is designed to be easy to use and incredibly versatile, but it is not as fast. The reason for this is that MATLAB code is a sort of facade code, which runs another underlying code, causing the program to run much more slowly. Programming in C code would allay this problem, allowing many more lines of data and different stocks to be run more quickly than with MATLAB. Programming in C would also require the use of MCMC minimization since fmincon and fminsearch are not available in C. This minimization, while more complex, is a robust method and is less prone to finding local minima in the place of the global minimum than fminsearch and fmincon.

Another improvement which can be made to the current research is to employ multiple years of stock data and multiple stocks to allow further validation of these results.

# ACKNOWLEDGEMENTS

The author would like to take this opportunity to thank Dr. Zhiguang Wang of the SDSU Department of Economics for introducing him to this subject, supplying the papers with the models and raw data, and for his many hours of mentorship and guidance through this research. The author would also like to thank Dr. Jung-Han Kimn of the SDSU Department of Mathematics & Statistics for his relentless support and expertise.

# REFERENCES

[1] Bakshi, G., Cao, C., and Chen, Z. *Empirical Performance of Alternative Option Pricing Models.*The Journal of Finance.v52 No.5 Dec 1997, 2003-2049.

[2] Duffie, D., Pan, J., and Singleton, K. *Transform Analysis and Asset Pricing for Affine Jump-Diffusions.*Econometrica.v68 (6 Nov) 2000, 1343-1376.

[3] Kythe, P., and Schäferkotter, M. *Handbook of Computational Methods for Integration*. Boca Raton, FL: Chapman & Hall/CRC, 2005.