Fast Monte Carlo Valuation of American Options under Stochastic Volatility and Interest Rates

Y. Hilpisch*

August 2011

Abstract

This article analyzes the valuation of American options by Monte Carlo simulation in the presence of stochastic volatility and interest rates. For practical applications, it is important to have available efficient, i.e. accurate and fast, valuation algorithms in such a context. Recently, Medvedev and Scaillet (2009) introduced new value approximation techniques whose technical implementation takes less than one second per American option valuation. They state that the valuation by Monte Carlo simulation takes "dozens of minutes" in comparison. We show that this does not have to be the case: our `Python` implementation of the Least-Squares Monte Carlo (LSM) algorithm with control variates takes only less than one second for an accurateness that is consistent with typical option bid-ask spreads and tick sizes.

Contents

*Dr. Yves J. Hilpisch, Managing Director, Visixion GmbH, Rathausstrasse 75-79, 66333 Voelklingen, Germany. Email contact@visixion.com, Internet `www.visixion.com`. I am grateful to Michael Schwed who helped improve my first `Python` codings. This version replaces a number of earlier drafts.

1 Introduction

Monte Carlo simulation (MCS) is among the most important numerical algorithms of the 20th century (cf. Cipra (2000)). Its importance for financial applications stems from the fact that it is most flexible in terms of financial products that can be valued. However, it took until the 21st century that the problem of valuing American options by MCS was satisfactorily solved by Longstaff and Schwartz (2001). They propose a least-squares regression algorithm to estimate the continuation value of an option step-by-step by backwards induction.[1]

Although quite flexible, MCS is generally not very fast since millions of computations are necessary to value a single option. Consider a simulation run with 100 time intervals (= 100 exercise dates) and 100,000 paths for an American put option on a single stock with constant volatility and constant short rate. You need 10 million random numbers, several arrays of size 101 times 100,000 and you have to estimate 100 least-squares regressions over 100,000 numbers as well as to discount 100 times 100,000 numbers.

If you enrich the financial model to include, for example, stochastic volatility and stochastic interest rates the number of necessary calculations further increases substantially. That is why researchers try to come up with closed-form valuation formulas that take only fractions of a second to evaluate numerically, like the celebrated Black-Scholes formula. In this regard, Medvedev and Scaillet (2009)—abbreviated in the following by MS2009—derive approximations for American option values under stochastic volatility (of Heston (1993) type) and stochastic interest rates (of Cox, Ingersoll and Ross (1985) type) which can be evaluated very fast. They write on page 16:

> "To give an idea of the computational advantage of our method, a Matlab code implementing the algorithm of Longstaff and Schwartz (2001) takes dozens of minutes to compute a single option price while our approximation takes roughly a tenth of a second."

[1] Cf. Glasserman (2004) for a comprehensive introduction into Monte Carlo methods for financial engineering. Cf. Kohler (2009) for a survey of regression-based valuation approaches for American options.

In this article, our aim is not to improve upon their approximation techniques. Our aim is rather to demonstrate that MCS does not necessarily take as long as *dozens of minutes* to come up with reasonably accurate value estimates—but generally less than a second.

Section 2 introduces the financial model of MS2009 in continuous time, provides an approximate discretization for simulation and outlines the general approach to value American options in such a setting by Least-Squares Monte Carlo (LSM). To improve upon accuracy of the LSM algorithm we use European options as control variates. Therefore, section 2 also addresses the valuation of European options by Fourier transform methods. Section 3 discusses practical accuracy issues and, in this light, numerical results from a `Python` script. The script implements the LSM algorithm for American put options with European put options as control variates. It contains also other features like moment matching techniques and antithetic paths. Section 4 concludes.

2 Financial Model

2.1 Continuous Time Version

2.1.1 Model Economy

Given is a filtered probability space $\{\Omega, \mathcal{F}, \mathbb{F}, P\}$ representing uncertainty in the model economy with final date T where $0 < T < \infty$. Ω denotes the continuous state space, $\mathcal{F}$ a $\sigma-$algebra, $\mathbb{F}$ a filtration—i.e. a family of increasing $\sigma-$algebras $\mathbb{F} \equiv \{\mathcal{F}_{t\in[0,T]}\}$ with $\mathcal{F}_0 \equiv \{\emptyset, \Omega\}$ and $\mathcal{F}_T \equiv \mathcal{F}$—and P the real or objective probability measure.

MS2009 consider a combination of the stochastic volatility model of Heston (1993, H93) and the stochastic interest rate model of Cox, Ingersoll and Ross (1985, CIR85). The stochastic volatility index part of the model in risk-neutral form is given by the following stochastic differential equations (SDE)

$$dS_t = r_t S_t dt + \sqrt{v_t} S_t dZ_t^1 \tag{1}$$

$$dv_t = \kappa_v(\theta_v - v_t)dt + \sigma_v \sqrt{v_t} dZ_t^2 \tag{2}$$

for $0 \leq t \leq T$ and with the following meaning of the variables and parameters

- S_t index level at date t
- r_t risk-less short rate at date t
- v_t variance at date t
- κ_v speed of adjustment of v_t to ...
- ... θ_v, the long-term average of the variance
- σ_v volatility coefficient of the index's variance
- Z_t^n standard Brownian motions

Similarly, the stochastic short rate part of the model is given by the SDE

$$dr_t = \kappa_r(\theta_r - r_t)dt + \sigma_r \sqrt{r_t} dZ_t^3 \tag{3}$$

where the variables and parameters are defined as

- r_t short rate at date t
- κ_r speed of adjustment of r_t to ...
- ... θ_r, the long-term average of the short rate
- σ_r volatility coefficient of the short rate
- Z_t^3 standard Brownian motion

All stochastic processes are adapted to the filtration $\mathbb{F}$. Moreover, assume that correlations are given by $dZ_t^1 dZ_t^2 \equiv \rho dt$, $dZ_t^1 dZ_t^3 \equiv dZ_t^2 dZ_t^3 \equiv 0$, i.e. the short rate process is neither correlated with the index level nor the variance. The time t value of a zero-coupon bond paying one unit of currency at $T, 0 \leq t < T$, is

$$B_t(T) = \mathbf{E}_t^Q \left(\exp \left(- \int_t^T r_u du \right) \right)$$

with $\mathbf{E}$ being the expectation operator and Q a risk-neutral probability measure equivalent to the real world measure P. $\mathbf{E}_t(\cdot)$ is short for the conditional expectation $\mathbf{E}(\cdot|\mathcal{F}_t)$.

We define the set of uncertainties by $X_t \equiv (S_t, v_t, r_t)$. By the Fundamental Theorem of Asset Pricing, the time t value of an attainable[2], $\mathcal{F}_T-$measurable contingent claim $V_T \equiv h_T(X_T) \geq 0$ (satisfying suitable integrability conditions) is given by arbitrage as

$$V_t = \mathbf{E}_t^Q \left(B_t(T) V_T \right)$$

with $V_0 = \mathbf{E}_0^Q \left(B_0(T) V_T \right)$ as the important special case for valuation purposes. The contingent claim could be a European call option maturing at T with payoff $h_T(X_T) \equiv \max[S_T - K, 0]$. It could also be a European put with payoff $h_T(X_T) \equiv \max[K - S_T, 0]$. In both cases, K is the fixed strike price of the option.

The valuation of contingent claims with American exercise, like American put options, is more involved. This problem can be formulated as an optimal stopping problem of the form

$$V_0 = \sup_{\tau \in [0,T]} \mathbf{E}_0^Q (B_0(\tau) h_\tau(X_\tau)) \tag{4}$$

with V_0 being the present value of the American derivative, τ a $\mathbb{F}-$adapted stopping time, T the date of maturity, $B_0(\tau)$ the discount factor appropriate for stopping time τ, h_τ a non-negative, $\mathcal{F}_\tau-$measurable payoff function and X_τ the market model vector process stopped at time $t = \tau$. The expectation is again taken under an equivalent risk-neutral measure Q.

2.1.2 European Options

In this article, we focus—along the lines of MS2009—on American put options. To improve accuracy of the Monte Carlo simulation and the LSM algorithm, we apply among others control variates. The fundamental idea behind control variates is that, in addition to the desired estimator (see equation (15) below), a second Monte Carlo estimator is derived

[2] A contingent claim is *attainable* if it can be replicated via an admissible trading strategy in available securities and contingent claims—a trading strategy is admissible if it is predictable and self-financing and if its value is at all times bounded from below. In the case of the financial model of this article this could imply that trading in at least one other contingent claim, in addition to the index and zero-coupon bonds with suitable maturity, must be possible. Cf. Gatheral (2006, ch. 2).

during the same simulation. However, what distinguishes the second estimator from the desired one is that its true value is known exactly. Since for the second estimator, i.e. the control variate, the error is known one can correct the desired estimator by an appropriate amount—generally based on the correlation between the respective simulated values for the two target quantities, i.e. the simulated American put option values and the control variate's simulated values at time $t = 0$.

It is well-known that the higher the correlation between a specific variate and the desired estimator the better the correction results one can expect. For the setting of the current article, the European counterpart of the American put option is a natural candidate since their values are highly correlated under certain circumstances.[3]

Fortunately, the values of European calls and puts in the model of MS2009 are known in semi-analytic form by Fourier transform techniques and characteristic functions. Since the short rate part of the model is not correlated with the stochastic volatility index part, it suffices to consider the characteristic function ϕ_0^{H93} of the Heston (1993) model (see equation (16) in appendix A). Equipped with the characteristic function, the following Fourier-based formula of Lewis (2001) to price European call options with strike K and maturity date T can be applied

$$C_0^{H93}(K,T) = S_0 - B_0(T)\frac{\sqrt{S_0 K}}{\pi}\int_0^\infty \mathbf{Re}\left[e^{-iuk}\phi_0^{H93}(u - i/2, T)\right]\frac{du}{u^2 + 1/4} \tag{5}$$

where $k \equiv \log(S_0/K)$. The price of the corresponding put follows from put-call-parity as

$$P_0^{H93}(K,T) = C_0^{H93}(K,T) + B_0(T)K - S_0 \tag{6}$$

The discount factor $B_0(T)$ is known in closed form for the CIR85 model. Appendix B provides the respective formula.

Appendix C contains a `Python` script implementing the pricing approach for European options in the H93 model and the formula for the discount factor $B_0(T)$ in the CIR85 model.

2.2 Discrete Time Version

2.2.1 Discretization of Processes

To simulate the financial model, i.e. to generate numerical values for X_t, it has to be discretized. To this end, divide the given time interval $[0, T]$ in equidistant sub-intervals Δt such that now $t \in \{0, \Delta t, 2\Delta t, ..., T\}$, i.e. there are $M + 1$ points in time with $M \equiv T/\Delta t$. A discretization of the continuous time market model (1), (2) and (3) of MS2009 is then given by (with $s = t - \Delta t$)

$$\begin{aligned} S_t &= S_s \exp\left((\bar{r}_t - v_t/2)\Delta t + \sqrt{v_t}\sqrt{\Delta t} z_t^1\right) \\ \tilde{v}_t &= \tilde{v}_s + \kappa_v(\theta_v - \tilde{v}_s^+)\Delta t + \sigma_v\sqrt{\tilde{v}_s^+}\sqrt{\Delta t} z_t^2 \end{aligned} \tag{7}$$

[3]Correlation mainly depends on the moneyness of the options, i.e. correlation increases generally the more the options are out-of-the-money. The more the options are in-the-money the more important becomes the early exercise feature of American options and the lesser the correlation between American options and their European counterparts.

$$
\begin{aligned}
v_t &= \tilde{v}_t^+ && (8)\\
\tilde{r}_t &= \tilde{r}_s + \kappa_r(\theta_r - \tilde{r}_s^+)\Delta t + \sigma_r\sqrt{\tilde{r}_s^+}\sqrt{\Delta t}z_t^3 \\
r_t &= \tilde{r}_t^+ && (9)
\end{aligned}
$$

for $t \in \{\Delta t, ..., T\}$ with $\bar{r}_t \equiv (r_t + r_s)/2$ and the z_t^n being standard normally distributed. x^+ is notation for $\max[x, 0]$. The z_t^1 and z_t^2 are correlated with ρ while all other random variables are uncorrelated.

This discretization scheme is an Euler discretization and is generally called *full trunction* scheme.[4] The exact transition law of the square-root diffusions (2) and (3) is also known. Broadie and Kaya (2006) provide an in-depth analysis of this topic. However, when using the exact law there arises the problem of correlationg the increments of the processes correctly. Using Euler discretizations only, there are only normally distributed random numbers to be correlated for Monte Carlo simulation.[5]

2.2.2 Discrete Optimal Stopping

To value American options, the optimal stopping problem (4) also has to be discretized (cf. Kohler (2009, 2)):

$$V_0 = \sup_{\tau \in \{0, \Delta t, 2\Delta t, ..., T\}} \mathbf{E}_0^Q(B_0(\tau)h_\tau(X_\tau)) \tag{10}$$

The continuation value C_t at date t of the option, i.e. the value of not exercising the option at this date, is given under risk-neutrality as

$$C_t(x) = \mathbf{E}_t^Q(e^{-\bar{r}_{t+\Delta t}\Delta t}V_{t+\Delta t}(X_{t+\Delta t})|X_t = x) \tag{11}$$

using the Markow property of X_t. By a well-known result (cf. Kohler (2009, 6)) the value of the option at date t is then

$$V_t(x) = \max[h_t(x), C_t(x)] \tag{12}$$

i.e. the maximum of the payoff $h_t(x)$ of immediate exercise and the expected discounted payoff $C_t(x)$ of not exercising. Here, the major insight of Longstaff and Schwartz (2001) is applied to estimate the continuation value $C_t(x)$ by ordinary least-squares regression.

To begin with, consider the simulation of I paths of the three-dimensional market model process $X_t = (S_t, v_t, r_t)$ with results $X_{t,i}$ where $t \in \{0, \Delta t, ..., T\}$ and $i \in \{1, ..., I\}$. Longstaff and Schwartz (2001) propose to regress the I continuation values $Y_{t,i} \equiv e^{-\bar{r}_{t+\Delta t}\Delta t}V_{t+\Delta t,i}$ against the I simulated values for $X_{t,i}$. Given D basis functions b with $b_1, ..., b_D : \mathbf{R}^{3\times D} \to \mathbf{R}$ for the regression, the continuation value $C_{t,i}$ is according to their approach approximated by

$$\hat{C}_{t,i} = \sum_{d=1}^{D} \alpha_{d,t}^* \cdot b_d(X_{t,i}) \tag{13}$$

The optimal regression parameters $\alpha_{d,t}^*$ are the result of the minimization

[4] Cf. Lord, Koekkoek and van Dijk (2008) for an analysis of this and other biased discretization schemes for the square-root diffusion process.

[5] I am thankful to participants of a research colloquium in May 2011 at Saarland University for helpful discussions with respect to these issues.

$$\min_{\alpha_{1,t},...,\alpha_{D,t}} \frac{1}{I} \sum_{i=1}^{I} \left(Y_{t,i} - \sum_{d=1}^{D} \alpha_{d,t} \cdot b_d(X_{t,i}) \right)^2 \tag{14}$$

In some circumstances, the quality of the regression can be improved upon when restricting the paths involved in the regression to those where the option is in the money.

To apply the LSM, implement under risk-neutrality the following recipe:

- simulate I paths of X_t with $M+1$ time steps leading to values $X_{t,i}, t \in \{0, ..., T\}, i \in \{1, ..., I\}$
- for $t = T$ the option value is $V_{T,i} = h_T(X_{T,i})$ by arbitrage
- start iterating backwards $t = T - \Delta t, ..., 0$:
  - regress the $Y_{t,i}$ against the $X_{t,i}, i \in \{1, ..., I\}$, given D basis functions b
  - approximate $C_{t,i}$ by $\hat{C}_{t,i}$ according to (13) given the optimal parameters $\alpha^*_{d,t}$ from (14)
  - according to (12) set

$$V_{t,i} = \begin{cases} h_t(X_{t,i}) & \text{if } h_t(X_{t,i}) > \hat{C}_{t,i} \quad \text{(exercise takes place)} \\ Y_{t,i} & \text{if } h_t(X_{t,i}) \leq \hat{C}_{t,i} \quad \text{(no exercise takes place)} \end{cases}$$

  repeat iteration steps until $t = 0$
- for $t = 0$ calculate the LSM estimator

$$\hat{V}_0^{LSM} = \frac{1}{I} \sum_{i=1}^{I} V_{0,i} \tag{15}$$

Note that when updating option values $V_{t,i}$, the real continuation value $Y_{t,i}$ is to be taken and not the estimated one $\hat{C}_{t,i}$.

It is well-known that the LSM estimator (15)—for big enough I—provides a lower bound for the option's value. However, small I may lead to an in-sample bias of the regressions in the sense that the resulting exercise policy is better-than-optimal.[6] Since we will work with rather big I, this bias is neglected in the following, i.e. we use only one sample of paths for both the regressions and the valuation.

We now correct the estimator by the simulated differences gained from a control variate. Consider that we have simulated I paths of X_t to value an American put option with maturity T and strike K. Then there are also I simulated present values of the corresponding European put option. They are given by $P_{0,i} = B_0(T) h_T(S_{T,i}), i \in \{1, ..., I\}$, with $h_T(s) \equiv \max[K - s, 0]$. We correct the estimator (15) as follows

$$\hat{V}_0^{CV} = \frac{1}{I} \sum_{i=1}^{I} \left(V_{0,i} - \lambda \cdot (P_{0,i} - P_0^{H93}) \right)$$

For simplicity, we set $\lambda \equiv 1$. In fact, the results from a number of numerical experiments imply that this assumption yields generally better results than, for example, the statistical correlation which would be smaller than one in any case.

[6] For example, in the extreme case of $I = 1$ you have perfect foresight.

3 Numerical Results

3.1 Parameterized Financial Model

We consider all model parameterizations for the H93 and the CIR85 parts of the financial model from table 3 in MS2009. These are four different parameter sets for the financial model.

Per parameter set, American put options for three different maturities and moneyness levels, respectively, are valued:

- $T \in \left\{\frac{1}{12}, \frac{1}{4}, \frac{1}{2}\right\}$
- $K \in \{90, 100, 110\}$

All parameter sets and all values for the single options (for a total of 36 option values) are included in the `Python` script provided in appendix D. The script implements the LSM algorithm with certain options to alter algorithm features (like control variates, moment matching or antithetic paths).

3.2 Practical Accuracy Considerations

The following sub-section discusses a sample output of the `Python` script found in appendix D. In principle, the script generates LSM value estimates, given the model and simulation parameterizations, and compares these with the benchmark values from MS2009. As benchmark values we take their LSM values obtained by simulations with 50 exercise dates, 500 time steps and 1,000,000 paths.

Speed alone does not suffice in general to judge a financial algorithm and its technical implementation. In contrast, accuracy generally ranks first—in particular when it comes to pricing tasks based upon which important decisions are taken (e.g. in investing, trading, hedging). It is known, according to results from Clément, Lamberton and Protter (2002), that the LSM estimator converges to the true (theoretical) option value when the number of basis functions in the regression and the number of paths in the simulation goes to infinity. However, in practice you have to decide how closely you want to get to the true value, weighing accuracy against computational effort and speed.

How do we judge accuracy? As performance yardsticks we take the absolute and relative value difference between the script's output value and the MCS value of the American put options as reported in MS2009. We say that our value estimate is *reasonably accurate* if either the absolute difference is

- smaller than PY_1 *currency units* or
- smaller than PY_2 *percent*.

The first yardstick should apply to options with very small values (where the absolute difference is better suited) while the second rather applies to those with higher values (where the relative difference is better suited). However, the question arises how to quantify PY_1 and PY_2. To begin with, in practice there is not that single true option quote. There are, among others, *opening* and *closing* quotes as well as *bid* quotes and *ask* quotes during the trading day—the difference between bid and ask quotes generally called *spread*.

Table 1 reports average option quote spreads for put options on stocks in the Dow Jones Industrial Average (DJIA) index for the period 1996 to 2010. For the total sample of

Table 1: Option quote spreads for put options on stocks of the DJIA index[a]

Category	Type	Number	Maturity	Mid-Price	Spread	Rel. Spread
All	All	1,105,028	96.07	5.093	0.229	7.80%
Short	OTM	158,486	44.55	1.339	0.148	15.98%
Short	ATM	120,257	44.63	3.443	0.204	7.12%
Short	ITM	146,979	43.86	6.858	0.279	4.91%
Long	OTM	267,847	128.80	2.238	0.172	10.26%
Long	ATM	201,100	129.33	5.769	0.255	5.18%
Long	ITM	210,359	127.34	10.621	0.317	3.50%

[a]Data for the period 1996–2010; OTM, ATM, ITM = out-of-the, at-the, in-the-money options; number = number of put options included in the sample; maturity = average option maturity in days; mid-price = middle of bid and ask quotes in USD; spread = USD difference of bid and ask quote; relative spread = spread relative to mid-price. Source: Chaudhury (2011).

more than 1.1 mn options, the average spread is 0.229 USD or 7.8% relative to the average mid-price. These values vary with maturity of the put options and moneyness levels. The *smallest absolute spread* with 0.148 USD is observed for out-of-the-money options with short maturity. The *smallest relative spread* emerges with 3.5% for in-the-money options with long maturity.

To measure accuracy we consider the absolute difference between our script's values and the benchmark values from MS2009. It would therefore be reasonable to take half of the absolute and relative spreads observed at maximum. We therefore set the performance yardsticks to $PY_1 \equiv 0.025$ currency units (i.e. 2.5 cents[7]) and $PY_2 \equiv 0.015$ (i.e. 1.5%). In both cases, we take respectively much less than half of the smallest absolute and relative option quote spreads from table 1.

We say that the value estimate is reasonably accurate if *either* PY_1 *or* PY_2 is satisfied. We have to take this either-or approach since the put option values in MS2009 are partly so small that they would either not be quoted in practice (one option value is reported with 0.01 cents) or are only about the typical tick size (four option values are reported below 8 cents).

3.3 Results from Simulation

A numerical experiment with 10 simulation runs—for a total of 360 American put option values—yielded the following results:

```
Overall Statistics
----------------------------------------------------
Start Calculations        2011-08-04 11:49:23.187414
----------------------------------------------------
Name of Simulation          Base_5_20_35_TTF_2.5_1.5
Seed Value for RNG                            100000
Number of Runs                                     5
Time Steps                                        20
Paths                                          35000
```

[7]This is also half the tick size—i.e. the minimum allowed change of the price of an option—in table 1 for options with bid quotes below 3 USD. For options with bid quotes above 3 USD the minimum tick size is 10 cents.

```
Control Variates                              True
Moment Matching                               True
Antithetic Paths                             False

Option Prices                                  180
Absolute Tolerance                          0.0250
Relative Tolerance                          0.0150
Errors                                           0
Error Ratio                                 0.0000

Aver Val Error                             -0.0071
Aver Abs Val Error                          0.0144

Aver Rel Error                              3.8247
Aver Abs Rel Error                          4.3511

Time in Seconds                           127.1616
Time in Minutes                             2.1194
Time per Option                             0.7065
-------------------------------------------------
End   Calculations        2011-08-04 11:51:30.348976
-------------------------------------------------
```

All American put options are valued accurately given our yardsticks. The average valuation error was -0.71 cents and therewith less than 1 cent in absolute value. The average relative error is not quite representative since the relative error for option values of about 0.01 cents easily reaches 100% and more. Nevertheless, it is only about +3.8%. Average time per option is about 0.7 seconds—which has to be compared with the "dozens of minutes" reported in MS2009. Our approach seems to be 1,000 times faster (if we assume a single dozen of minutes) with an accurateness that is consistent with a typical market microstructure.

3.4 Interpretation of Results

What are the reasons for the combination of reasonable accuracy and valuation speed of the `Python` script? Actually, there is a number of reasons—some of which we will test in the next subsection:

- **implementation**: the LSM algorithm has been implemented in `Python` using the fast numerical library `Numpy` which runs at the speed of C code; this may be faster than `Matlab` or other environments like `R`
- **discretization**: we use the full trunction discretization scheme which is known to provide good numerical results in comparison to other biased schemes; we let the simulated index level paths according to (7) drift step-by-step by the average of the two relevant short rate values
- **control variates**: the use of European put options as control variates (cf. Glasserman (2004, sec. 4.1)) is of high importance for variance reduction and accuracy of the LSM estimator
- **moment matching**: we correct the set of standard normal pseudo-random numbers generated by `Python` to match the first two moments correctly (cf. Glasserman (2004, sec. 4.5)), i.e. the mean is adjusted to 0.0 and the standard deviation to 1.0; we also correct the first moment of the simulated index level paths according to (7) step-by-step to account for some remaining errors

- **antithetic paths**: as a general variance reduction technique we generate, as in MS2009, antithetic paths (cf. Glasserman (2004, sec. 4.2)) such that convergence of the algorithm may somewhat improve
- **use of paths**: we use only in-the-money paths such that both the estimation of the regressions becomes faster (in particular for out-of-the-money options) and convergence of the algorithm may improve
- **normalization**: to allow for a more accurate path generation based on the discrete financial model and to improve regression performance we normalize the initial index level S_0 to 1.0 and adjust payoffs accordingly
- **basis functions**: we use all in all ten different basis functions for the regessions in the LSM implementation
- **exercise at $t = 0$**: we allow for exercise at $t = 0$ such that we get at least the inner value as the option price for the in-the-money cases
- **paths**: our LSM implementation allows a significant reduction in the number of discretization intervals (20 instead of 500 as in MS2009) and paths (35,000 instead of 1,000,000); our approach reduces the number of necessary simulated values by a factor of more than 500 and more than halves the number of regressions (20 exercise dates instead of 50)
- **recycling**: we use the same set of random numbers for the 36 options to be valued per simulation run; we also use the same simulated processes for each of the three options per time-to-maturity
- **hardware**: of course, hardware also plays a role; the computational times reported for the script are from an Acer notebook with Intel Core 2 Duo processor T7500 (2.2 GHz, 800 MHz FSB, 4 MB L2 cache) and 2 GB DDR2 RAM; `Python 2.7` and `Numpy` ran on an Unbuntu Linux operation system—frankly, not the fastest environment one can imagine—so better hardware alone could further speed up calculations considerably

3.5 Importance of Algorithm Features

In this subsection, we report further simulation results for variants of the LSM algorithm implementation. The aim is to identify those features of the implementation that indeed contribute to accuracy. Using the same seed value for the Python pseudo-random number generator, we replicate the 180 American option valuations several times—changing respectively features of the algorithm implementation. Table 2 shows the results.

It is obvious that the use of control variates is of paramount importance for the accuracy. By contrast, moment matching and antithetic variates may be beneficial or not (if at all, then on a small scale). In view of the rather small additional computational time needed to include control variates they should be used whenever possible in such a context.

3.6 Higher Accuracy vs. Lower Speed

In some circumstances, our yardsticks used to assess accuracy may be too lax. Even if only for theoretical interest, one might be interested in the LSM estimator being closer to the true (theoretical) option value. To this end, we set the performance yardsticks now to $PY_1 \equiv 0.01$ currency units (i.e. 1 cent) and $PY_2 \equiv 0.01$ (i.e. 1%). In particular, the 1 cent

Table 2: Simulation results for different configurations of the LSM algorithm and an accuracy level of $PY_1 = 0.025$ and $PY_2 = 0.015$.[a]

R	M	I	**CV**	**MM**	**AP**	**ATol**	**RTol**	**#Op**	**Err**	**AvEr**	**Sec/O.**
5	15	25000	True	True	True	0.025	0.015	180	1	-0.01	0.402
5	15	35000	True	True	True	0.025	0.015	180	3	-0.012	0.54
5	20	25000	True	True	True	0.025	0.015	180	1	-0.003	0.507
5	20	35000	True	True	True	0.025	0.015	180	1	-0.012	0.699
5	25	25000	True	True	True	0.025	0.015	180	2	-0.002	0.618
5	25	35000	True	True	True	0.025	0.015	180	0	-0.008	0.894
5	20	35000	True	True	False	0.025	0.015	180	0	-0.007	0.815
5	25	35000	True	True	False	0.025	0.015	180	1	-0.007	1.012
5	20	35000	True	False	False	0.025	0.015	180	2	-0.006	0.67
5	25	35000	True	False	False	0.025	0.015	180	2	-0.006	0.963
5	20	35000	False	False	False	0.025	0.015	180	53	-0.019	0.737
5	25	35000	False	False	False	0.025	0.015	180	45	-0.013	1.262

[a] R = number of runs, M = number of time intervals, I = number of simulation paths, CV = control variates, MM = moment matching, AP = antithetic paths, ATol = absolute performance yardstick, RTol = relative performance yardstick, #Op = number of options, Err = number of errors, AvEr = average error in currency units, Sec/O. = seconds per option valuation.

threshold is reasonable since it represents the smallest currency unit in general. Therefore it is often used to judge accuracy. For example, Longstaff and Schwartz (2001, 127) write: "Of the 20 differences shown in Table 1, 16 are less than or equal to one cent in absolute value."

Table 3: Simulation results for different configurations of the LSM algorithm and an accuracy level of $PY_1 = 0.01$ and $PY_2 = 0.01$.[a]

R	M	I	**CV**	**MM**	**AP**	**ATol**	**RTol**	**#Op**	**Err**	**AvEr**	**Sec/O.**
5	20	35000	True	True	False	0.01	0.01	180	13	-0.007	0.717
5	25	50000	True	True	False	0.01	0.01	180	11	-0.008	1.507
5	25	75000	True	True	True	0.01	0.01	180	13	-0.011	2.096
5	25	75000	True	True	False	0.01	0.01	180	10	-0.009	2.993
5	50	50000	True	True	False	0.01	0.01	180	11	-0.007	3.231
5	50	75000	True	True	False	0.01	0.01	180	7	-0.006	4.935
5	50	100000	True	True	False	0.01	0.01	180	4	-0.007	6.044
5	75	100000	True	True	False	0.01	0.01	180	8	-0.006	8.696

[a] Notation compare table 2.

To better meet the new yardsticks, we increase the number of time intervals to 50 and the number of paths to 100,000. As the results in table 3 show, there are four valuation errors for the 180 options. The average valuation error of -0.007 cents is below one cent in absolute value. Time per option valuation increases to a bit more than 6 seconds. However, further increasing the number of time intervals to 75 ceteris paribus does not guarantee

better valaution results, as is also illustrated in table 3.

These results illustrate the trade-off between valuation accuracy and speed quite well. By increasing the number of time intervals and paths per simulation, you can get closer and closer to the true (theoretical) value—just as the convergence results of Clément, Lamberton and Protter (2002) imply. Longer valuation times are the price to pay.

4 Conclusions

Of course, Monte Carlo simulation in general and the LSM algorithm in particular are computationally demanding approaches and therefore often much slower than alternative valuation techniques—provided alternatives exist at all. However, the computational disadvantage of LSM can be mitigated by using problem specific information, like control variates, or general variance reduction techniques, like moment matching. Our `Python` script takes 0.7 seconds for one American put option with an accuracy of either 2.5 cents or 1.5 percent and better. This is more than 1,000 times faster than the times reported in MS2009. Increasing accuracy to 1 cent and 1 percent, respectively, makes it necessary to use more paths per simulation which leads to valuation times of about 6 seconds (for an error ratio of 2.2%). In view of the LSM's flexibility—see the diverse examples in Longstaff and Schwartz (2001)—a computational time of a few seconds per option seems acceptable for most applications.

There is probably further potential to reduce computational times per American option value. Our focus in this article is to show that the valuation of American options can be done two to three orders of magnitude faster than stated in MS2009. Therefore the script in appendix D has not really been optimized. However, it is questionable that—assuming the same hardware—one could reduce the time by another order of magnitude to less than 0.1 seconds, for example. But maybe one could save a further 20–30%. Beyond that one would probably need better hardware or need to redesign the LSM algorithm completely to allow for a parallel implementation approach.[8]

Our valuation speeds are per se interesting when compared with the benchmark of MS2009. However, they are even more impressing when taking into account that we reach these speeds with `Python`, an interpreted language. The reader is encouraged to play with the `Python` scripts provided in the appendix—which are self-contained and can be used immediately—to further explore features and potential improvements of our implementation.

[8]This is an approach followed, among others, by Choudhury et al. (2008). For a parallel architecure with 32 processor nodes, they report speed-ups of up to 18 times compared with an optimized serial code.

A Characteristic Function of Heston (1993) Model

The characteristic function ϕ_0^{H93} of the Heston stochastic volatility model is given by[9]

$$\phi_0^{H93}(u,T) = e^{H_1(u,T)+H_2(u,T)v_0} \tag{16}$$

with the following definitions

$$\begin{aligned}
c_1 &\equiv \kappa_v\theta_v \\
c_2 &\equiv -\sqrt{(\rho\sigma_v ui-\kappa_v)^2-\sigma_v^2(-ui-u^2)} \\
c_3 &\equiv \frac{\kappa_v-\rho\sigma_v ui+c_2}{\kappa_v-\rho\sigma_v ui-c_2} \\
H_1(u,T) &\equiv r_{0,T}uiT+\frac{c_1}{\sigma_v^2}\left\{(\kappa_v-\rho\sigma_v ui+c_2)T-2\log\left[\frac{1-c_1e^{c_3T}}{1-c_3}\right]\right\} \\
H_2(u,T) &\equiv \frac{\kappa_v-\rho\sigma_v ui+c_2}{\sigma_v^2}\left[\frac{1-e^{c_2T}}{1-c_3e^{c_2T}}\right]
\end{aligned}$$

and all variables as defined as in the main text. In H_1 we set $r_{0,T} = -\log(B_0(T))/T$.

B Discount Factor in Cox, Ingersoll, Ross (1985) Model

The discount factor for discounting cash flows due at time T to time $t = 0$, i.e. the present value of a zero-coupon bond paying one unit of currency at T, takes the form[10]

$$B_0(T) = b_1(T)e^{-b_2(T)r_0} \tag{17}$$

$$b_1(T) \equiv \left[\frac{2\gamma\exp(0.5(\kappa_r+\gamma)T)}{2\gamma+(\kappa_r+\gamma)(e^{\gamma T}-1)}\right]^{\frac{2\kappa_r\theta_r}{\sigma_r^2}} \tag{18}$$

$$b_2(T) \equiv \frac{2\left(e^{\gamma T}-1\right)}{2\gamma+(\kappa_r+\gamma)\left(e^{\gamma T}-1\right)} \tag{19}$$

$$\gamma \equiv \sqrt{\kappa_r^2+2\sigma_r^2} \tag{20}$$

The `Python` script of appendix C implements the formulas (17)–(20).

[9] Cf. Heston (1993) or Gatheral (2006, ch. 2).
[10] Cf. Glasserman (2004, 128-129).

C Python Script for European Option Valuation

In order to run the Python scripts provided in this appendix properly you should install the following:

- Python 2.6 (www.python.org)
- Numpy 1.3.x (http://numpy.scipy.org)
- SciPy 0.7.x (www.scipy.org)

```
#
# Valuation of European Call and Put Options
# Under Stochastic Volatility and Interest Rates
# Parameter Examples from Medvedev & Scaillet (2009):
# "Pricing American Options Under Stochastic Volatility
# and Stochastic Interest Rates"
# Working Paper No. 429, Finrisk --- MS (2009)
#
# (c) Visixion GmbH - Dr. Y. Hilpisch
# Script for illustration purposes only.
# August 2011
#
# SVSI_Euro_Valuation.py
#
from numpy import *
from scipy.integrate import *

#
# Example Parameters H93 Model
#
kappa_v = 1.5
theta_v = 0.02
sigma_v = 0.15
rho     = 0.1
v0      = 0.01
S0    = 100.0
K     = 90.0
T     = 1.0/12

#
# Valuation by Integration
#
def H93_Value_Call_Int(S0,K,T,B0T,kappa_v,theta_v,sigma_v,rho,v0):
    r=-log(B0T)/T
    Int = quad(lambda u:H93_Int_Func(u,S0,K,T,r,kappa_v,
                        theta_v,sigma_v,rho,v0),0,1000)[0]
    return max(0,S0-B0T*sqrt(S0*K)/pi*Int)

#
# Integration Function
#
def H93_Int_Func(u,S0,K,T,r,kappa_v,theta_v,sigma_v,rho,v0):
    CF = H93_Char_Func(u-1j*0.5,T,r,kappa_v,theta_v,sigma_v,rho,v0)
    return 1/(u**2+0.25)*(exp(1j*u*log(S0/K))*CF).real

#
# Characteristic Function
#
def H93_Char_Func(u,T,r,kappa_v,theta_v,sigma_v,rho,v0):
    c1 = kappa_v*theta_v
    c2 = -sqrt((rho*sigma_v*u*1j-kappa_v)**2-sigma_v**2*(-u*1j-u**2))
    c3 = (kappa_v-rho*sigma_v*u*1j+c2)/(kappa_v-rho*sigma_v*u*1j-c2)
    H1 = (r*u*1j*T+(c1/sigma_v**2)*((kappa_v-rho*sigma_v*u*1j+c2)*T
                        -2*log((1-c3*exp(c2*T))/(1-c3))))
    H2 = ((kappa_v-rho*sigma_v*u*1j+c2)/sigma_v**2*
```

```
        ((1-exp(c2*T))/(1-c3*exp(c2*T))))
    return exp(H1+H2*v0)

#
# Example Parameters CIR85 Model
#
kappa_r,theta_r,sigma_r,r0,T=0.3,0.04,0.1,0.04,1.0/12

#
# (Zero-Coupon-)Bond Valuation Formula
#
def gamma(kappa_r,sigma_r):
    return sqrt(kappa_r**2+2*sigma_r**2)

def b1(alpha):
    kappa_r,theta_r,sigma_r,r0,T = alpha
    g = gamma(kappa_r,sigma_r)
    return ((2*g*exp((kappa_r+g)*T/2))/
            (2*g+(kappa_r+g)*(exp(g*T)-1)))**(2*kappa_r*theta_r/sigma_r**2)

def b2(alpha):
    kappa_r,theta_r,sigma_r,r0,T = alpha
    g = gamma(kappa_r,sigma_r)
    return ((2*(exp(g*T)-1))/
            (2*g+(kappa_r+g)*(exp(g*T)-1)))

def B(alpha):
    b_1 = b1(alpha); b_2 = b2(alpha)
    kappa_r,theta_r,sigma_r,r0,T = alpha
    return b_1*exp(-b_2*r0)

#
# Example Values
#
B0T=B([kappa_r,theta_r,sigma_r,r0,T]) # Discount Factor
C0=H93_Value_Call_Int(S0,K,T,B0T,kappa_v,theta_v,sigma_v,rho,v0)
P0=C0+K*B0T-S0
print "H93 Call Value   ", C0
print "H93 Put  Value   ", P0
```

D Python Script for American Option Valuation

```
#
# Valuation of American Put Options
# Under Stochastic Volatility and Interest Rates
# Examples from Medvedev & Scaillet (2009):
# "Pricing American Options Under Stochastic Volatility
# and Stochastic Interest Rates"
# Working Paper No. 429, Finrisk --- MS (2009)
#
# (c) Visixion GmbH - Dr. Y. Hilpisch
# Script for illustration purposes only.
# August 2011
#
# SVSI_LSM_Con_Var.py
#
from numpy import *
from numpy.random import standard_normal,poisson,seed
from numpy.linalg import cholesky,lstsq
from datetime import *
from time import time
from SVSI_Euro_Valuation import *

t1=time()
```

```
d1=datetime.now()

# 'True' American Options Prices by Monte Carlo
# from MS (2009), table 3
trueList=array(((0.0001,1.0438,9.9950,0.0346,1.7379,9.9823,
                 0.2040,2.3951,9.9726),      # panel 1
                (0.0619,2.1306,10.0386,0.5303,3.4173,10.4271,
                 1.1824,4.4249,11.0224),     # panel 2
                (0.0592,2.1138,10.0372,0.4950,3.3478,10.3825,
                 1.0752,4.2732,10.8964),     # panel 3
                (0.0787,2.1277,10.0198,0.6012,3.4089,10.2512,
                 1.2896,4.4103,10.6988)))    # panel 4

# Cox, Ingersoll, Ross (1985) Parameters
# from MS (2009), table 3, panel 1
kappa_r=0.3
theta_r=0.04
sigma_r=0.1
r0     =0.04

# Heston (1993) Parameters
# from MS (2009), table 3
para=array(((0.01,1.5,0.15,0.1),    # panel 1
            # (v0,kappa_v,sigma_v,rho)
            (0.04,0.75,0.3,0.1),    # panel 2
            (0.04,1.5,0.3,0.1),     # panel 3
            (0.04,1.5,0.15,-0.5)))  # panel 4

theta_v=0.02    # Long Term Volatility Level
S0     =100.0   # Initial Index Level

# General Simulation Parameters
mL=[20]         # Time Steps
iL=[35000]      # Number of Paths per Valuation
tL=[1.0/12,0.25,0.5]   # Maturity List
kL=[90,100,110]         # Strike List
coVar=True      # Use of Control Variates
moMatch=True    # Random Number Correction (std + mean + drift)
antiPaths=False# Antithetic Paths for Variance Reduction
D=10            # Number of Basis Functions
PY1=0.025       # Performance Yardstick 1: Abs. Error in Currency Units
PY2=0.015       # Performance Yardstick 2: Rel. Error in Decimals
R=5             # Number of Simulation Runs
SEED=100000     # Seed Value

#
# Function for Short Rate and Volatility Processes
#
def eulerMRProc(x0,sigma,kappa,theta,row,CM):
    xh=zeros((M+1,I),'d')
    x=zeros((M+1,I),'d')
    xh[0,:]=x0;x[0,:]=x0
    for t in range(1,M+1):
        ran=dot(CM,rand[:,t,:])
        xh[t,:]+=xh[t-1,:]
        xh[t,:]+=kappa*(theta-maximum(0,xh[t-1,:]))*dt # Full Truncation
        xh[t,:]+=sqrt(maximum(0,xh[t-1,:]))*sigma*ran[row]*sqrt(dt)
        x[t,:]=maximum(0,xh[t,:])
    return x

#
# Function for Heston Index Process
#
def eulerSExp(r,S0,v,row,CM):
    S=zeros((M+1,I),'d')
    S[0,:]=S0
    bias=0.0
    for t in range(1,M+1,1):
```

```
            ran=dot(CM,rand[:,t,:])
            if moMatch==True:
                bias=mean(sqrt(v[t,:])*ran[row]*sqrt(dt))
            S[t,:]=S[t-1,:]*exp((((r[t,:]+r[t-1,:])/2-v[t,:]/2)*dt)+
                    sqrt(v[t,:])*ran[row]*sqrt(dt)-bias)
        return S

    def RNG(M,I):
        if antiPaths==True:
            randDummy=standard_normal((3,M+1,I/2))
            rand=concatenate((randDummy,-randDummy),2)
        else:
            rand=standard_normal((3,M+1,I))
        if moMatch==True:
            rand=rand/std(rand)
            rand=rand-mean(rand)
        return rand

    #
    # Valuation
    #
    t0=time()
    for M in mL:      # Number of Time Steps
        for I in iL: # Number of Paths
            t1=time();d1=datetime.now()
            absError=[];relError=[];l=0.0;errors=0
            # Name of the Simulation Setup
            name=('Base_'+str(R)+'_'+str(M)+'_'+str(I/1000)+'_'
                  +str(coVar)[0]+str(moMatch)[0]+str(antiPaths)[0]+
                  '_'+str(PY1*100)+'_'+str(PY2*100))
            seed(SEED)          # RNG seed value
            for i in range(R): # Simulation Runs
                print "\nSimulation Run %d of %d" %(i+1,R)
                print "-----------------------------------------------------"
                print "Elapsed Time in Minutes %8.2f" %((time()-t0)/60)
                print "-----------------------------------------------------"
                for panel in range(4): # Panels
                    # Correlation Matrix
                    v0,kappa_v,sigma_v,rho=para[panel,:]
                    CovMat=zeros((3,3),'d')
                    CovMat[0,:]=[1.0,rho,0.0]
                    CovMat[1,:]=[rho,1.0,0.0]
                    CovMat[2,:]=[0.0,0.0,1.0]
                    CM=cholesky(CovMat)
                    print "\n\n Results for Panel %d\n" %(panel+1)
                    print " v0=%3.2f, sigma_v=%3.2f, kappa_v=%3.2f, rho=%3.2f" \
                          %(v0,sigma_v,kappa_v,rho)
                    print " -----------------------------------------------------"
                    z=0
                    for T in tL: # Times-to-Maturity
                        B0T=B([kappa_r,theta_r,sigma_r,r0,T])
                            # Discount Factor B0(T)
                        r,v,S,h,V=0.0,0.0,0.0,0.0,0.0
                            # Memory Clean-up
                        dt=T/M
                            # Time Interval in Years
                        rand=RNG(M,I)
                            # Random Numbers
                        r=eulerMRProc(r0,sigma_r,kappa_r,theta_r,0,CM)
                            # Short Rate Process Paths
                        v=eulerMRProc(v0,sigma_v,kappa_v,theta_v,2,CM)
                            # Volatility Process Paths
                        S=eulerSExp(r,S0/S0,v,1,CM)
                            # Index Level Process Paths
                        print "\n  Results for Time-to-Maturity %6.3f" %T
                        print "  ---------------------------------------"
                        for K in kL: # Strikes
                            h,V=0.0,0.0           # Memory Clean-up
```

```
h=maximum(K/S0-S,0)   # Inner Value Matrix
V=maximum(K/S0-S,0)   # Value/Cash Flow Matrix
for t in range(M-1,0,-1):
    df=exp(-((r[t,:]+r[t+1,:])/2)*dt)
    dummy=greater(h[t,:],0)  # Select ITM Paths
    relevant=nonzero(dummy)
    relS=compress(dummy,S[t,:])
    p=len(relS)
    if p==0:
        cv=zeros((I),'d')
    else:
        relv=compress(dummy,v[t,:])
        relr=compress(dummy,r[t,:])
        relV=(compress(dummy,V[t+1,:])
                    *compress(dummy,df))
        matrix=zeros((D+1,p),'d')
        matrix[10,:]=relS*relv*relr
        matrix[9,:]=relS*relv
        matrix[8,:]=relS*relr
        matrix[7,:]=relv*relr
        matrix[6,:]=relS**2
        matrix[5,:]=relv**2
        matrix[4,:]=relr**2
        matrix[3,:]=relS
        matrix[2,:]=relv
        matrix[1,:]=relr
        matrix[0,:]=1
        pol=lstsq(matrix.transpose(),relV)
        cv=dot(pol[0],matrix)
    erg=zeros((I),'d')
    put(erg,relevant,cv)
    V[t,:]=where(h[t,:]>erg,h[t,:],V[t+1,:]*df)

df=exp(-((r[0,:]+r[1,:])/2)*dt)
## European Option Values
C0=H93_Value_Call_Int(S0,K,T,B0T,kappa_v,
                      theta_v,sigma_v,rho,v0)
P0=C0+K*B0T-S0
P0_MCS=B0T*sum(h[-1,:])/I*S0
## Determination of Correlation
x = B0T*h[-1,:]
y = V[1,:]*df
b = sum((x-mean(x))*(y-mean(y)))/sum((x-mean(x))**2)
## Control Variate Correction
if coVar==True:
    yCV=y-1.0*(B0T*h[-1,:]-P0/S0)
    # Set b instead of 1.0 to use stat. correlation
else:
    yCV=y
SE=std(yCV)/sqrt(I)*S0
A0CV=max(sum(yCV)/I*S0,h[0,0]) # LSM Control Variate
A0LS=max(sum(y)/I*S0,h[0,0])   # Pure LSM
## Errors
diff=A0CV-trueList[panel,z]
rdiff=diff/trueList[panel,z]
absError.append(diff)
relError.append(rdiff*100)
## Output
br = "    ----------------------------------"
print "\n  Results for Strike %4.0f\n" %K
print "    European Put Value MCS    %8.4f" % P0_MCS
print "    European Put Value Closed %8.4f" % P0
print "    American Put Value LSM    %8.4f" % A0LS
print "    Correlation               %8.4f" % b ,"\n",br
print "    American Put Value CV     %8.4f" % A0CV
print "    Standard Error LSM CV     %8.4f" % SE ,"\n",br
print "    American Put Value Paper  %8.4f" % trueList[panel,z]
print "    Valuation Error (abs)     %8.4f" % diff
```

```
                print "    Valuation Error (rel)     %8.4f" % rdiff
                if abs(diff)<PY1 or abs(diff)/trueList[panel,z]<PY2:
                        print "      Accuracy ok!\n"+br
                        CORR =True
                else:
                        print "      Accuracy NOT ok!\n"+br
                        CORR =False
                        errors=errors+1
                print "    %d Errors, %d Values, %.1f Min." \
                      %(errors,len(absError),float((time()-t1)/60))
                print "    %d Time Intervals, %d Paths" \
                      %(M,I)
                z=z+1;l=l+1
t2=time();d2=datetime.now()
br = "-------------------------------------------------------"
print "\n\nOverall Statistics","\n"+br
print "Start Calculations  %32s\n" % str(d1)+br
print "Name of Simulation  %32s" % name
print "Seed Value for RNG  %32d" % SEED
print "Number of Runs      %32d" % R
print "Time Steps          %32d" % M
print "Paths               %32d" % I
print "Control Variates    %32s" % coVar
print "Moment Matching     %32s" % moMatch
print "Antithetic Paths    %32s\n" % antiPaths
print "Option Prices       %32d" % l
print "Absolute Tolerance  %32.4f" % PY1
print "Relative Tolerance  %32.4f" % PY2
print "Errors              %32d" % errors
print "Error Ratio         %32.4f" % float(errors/l) +"\n"
print "Aver Val Error      %32.4f" % (sum(array(absError))/l)
print "Aver Abs Val Error  %32.4f\n" % (sum(abs(array(absError)))/l)
print "Aver Rel Error      %32.4f" % (sum(array(relError))/l)
print "Aver Abs Rel Error  %32.4f\n" % (sum(abs(array(relError)))/l)
print "Time in Seconds     %32.4f" % (t2-t1)
print "Time in Minutes     %32.4f" % ((t2-t1)/60)
print "Time per Option     %32.4f" %  float((t2-t1)/l)+"\n"+br
print "End   Calculations  %32s" %str(d2)+"\n"+br
```

References

[1] Broadie, Mark and Özgür Kaya (2006): "Exact Simulation of Stochastic Volatility and other Affine Jump Diffusion Processes." *Operations Research*, Vol. 54, No. 2, 217-231.

[2] Chaudhury, Mo (2011): "Option Bid-Ask Spread and Liquidity." *Working Paper*, McGill University, Desautels Faculty of Management, `www.ssrn.com`.

[3] Choudhury, Anamitra, Alan King, Sunil Kumar and Yogish Sabharwal (2008): "Optimizations in Financial Engineering: The Least-Squares Monte Carlo Method of Longstaff and Schwartz." in: *Proceedings of 2008 IEEE International Symposium on Parallel and Distributed Processing, 1-11.*

[4] Clément, Emmanuelle, Damien Lamberton and Philip Protter (2002): "An Analysis of a Least Squares Regression Algorithm for American Option Pricing." *Finance and Stochastics* , Vol. 17, 448-471.

[5] Cipra, Barry (2000): "The Best of the 20th Century: Editors Name Top 10 Algorithms." *SIAM News*, Vol. 33, No. 4, 1-2.

[6] Cox, John, Jonathan Ingersoll and Stephen Ross (1985): "A Theory of the Term Structure of Interest Rates." *Econometrica*, Vol. 53, 385-407.

[7] Gatheral, Jim (2006): *The Volatility Surface—A Practitioner's Guide.* John Wiley & Sons, Hoboken, New Jersey.

[8] Glasserman, Paul (2004): *Monte Carlo Methods in Financial Engineering.* Springer Verlag, New York et al.

[9] Heston, Steven (1993): "A Closed-From Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options." *The Review of Financial Studies*, Vol. 6, No. 2, 327-343.

[10] Kohler, Michael (2009): "A Review on Regression-based Monte Carlo Methods for Pricing American Options." *Working Paper*, Technical University of Darmstadt, Darmstadt, `www.mathematik.tu-darmstadt.de`.

[11] Lewis, Alan (2001): "A Simple Option Formula for General Jump-Diffusion and Other Exponential Lévy Processes." *Working Paper*, OptionCity, `www.optioncity.net`.

[12] Longstaff, Francis and Eduardo Schwartz (2001): "Valuing American Options by Simulation: A Simple Least-Squares Approach." *Review of Financial Studies*, Vol. 14, No. 1, 113-147.

[13] Lord, Roger, Remmert Koekkoek and Dick van Dijk (2008): "A Comparison of Biased Simulation Schemes for Stochastic Volatility Models." *Working Paper*, Tinbergen Institute, Amsterdam, `www.ssrn.com`.

[14] Medvedev, Alexey and Olivier Scaillet (2009): "Pricing American Options Under Stochastic Volatility and Stochastic Interest Rates." *Working Paper No. 429*, National Centre of Competence in Research—Financial Valuation and Risk Management, `www.ssrn.com`.