Risk Measures Extracted from Option Market Data Using Massively Parallel Computing

A Professional Master's Project
Submitted to the Faculty of the
WORCESTER POLYTECHNIC INSTITUTE
In partial fulfillment of the requirements for the
Professional Degree of
Master of Science
in
Financial Mathematics by

Min Zhao

April,2011

Approved:

Professor Domokos Vermes, Advisor

Professor Bogdan Vernescu, Head of Department

Abstract

The famous Black-Scholes formula provided the first mathematically sound mechanism to price financial options. It is based on the assumption, that daily random stock returns are identically normally distributed and hence stock prices follow a stochastic process with a constant volatility. Observed prices, at which options trade on the markets, don't fully support this hypothesis. Options corresponding to different strike prices trade as if they were driven by different volatilities.

To capture this so-called *volatility smile*, we need a more sophisticated option-pricing model assuming that the volatility itself is a random process. The price we have to pay for this stochastic volatility model is that such models are computationally extremely intensive to simulate and hence difficult to fit to observed market prices. This difficulty has severely limited the use of stochastic volatility models in the practice.

In this project we propose to overcome the obstacle of computational complexity by executing the simulations in a massively parallel fashion on the graphics processing unit (GPU) of the computer, utilizing its hundreds of parallel processors.

We succeed in generating the trillions of random numbers needed to fit a monthly options contract in 3 hours on a desktop computer with a Tesla GPU. This enables us to accurately price *any* derivative security based on the same underlying stock. In addition, our method also allows extracting quantitative measures of the riskiness of the underlying stock that are implied by the views of the forward-looking traders on the option markets.

Acknowledgement

With great pleasure, I would like to give special thanks to my advisor, Professor Domokos Vermes, for his guidance, support and encouragement during the 2 years of my graduate study at WPI. I also wish to thank Professor Marcel Blais and Professor Hasanjan Sayit, for all the great lectures they gave about financial mathematics.

Contents

1. Background

1.1 Volatility of risky investments

For risk-free investments, such as treasury bonds, we often describe its return using the following differential equation: $dB_t / B_t = rdt$. Here dB_t / B_t means the relative return of the bond during the infinitesimal period dt, and r is the interest rate.

For risky investments, like stocks, we need an extra term to model the uncertainty of their returns. The dynamics of the stock returns become the following stochastic differential equation: $dS_t / S_t = rdt + \sigma dW_t$. Here W_t refers to the standard Brownian motion, which is a stochastic process with continuous path and independent increment $W_t - W_s \sim N(0, t-s)$; σ is known as the volatility, it is defined as the standard deviation of the stock return over a certain period (often a year). Commonly, stock returns will fluctuate more with higher value of volatility.

1.2 European style options

A European call option is a financial contract between two parties: at a prescribed time in the future (known as the time of expiry T), the holder of the option has the right (but not the obligation) to purchase a underlying asset (like stock) at a prescribed amount (known as a the strike price K); while the writer of the contract has the potential obligation to sell the underlying asset if the holder wants to buy. The payoff of a European call option at expiry takes the form: $\max(S_T - K, 0)$.

A European put option has similar conditions as a European call, except that the holder has the right to sell the underlying asset to the writer at expiry at strike price. The payoff of a European put option at expiry takes the form:

$\max(K - S_T, 0)$.

"Moneyness" is defined to be the ratio of strike K over the current underlying price S_t. If the option could be exercised at the current time t for a positive payoff for the holder (i.e. $K < S_t$ for a call or $K > S_t$ for a put), then the option is called to be "in-the-money". The opposite case when an option is not to be exercised (i.e. $K > S_t$ for a call or $K < S_t$ for a put) is called to be "out-of-the-money". Out-of-the-money options are often traded by investors to hedge against large losses. Finally, "at-the-money" (ATM) refers to the case when the strike price is equal to the current underlying stock price (moneyness=1).

1.3 Black-Scholes-Merton model

Black-Scholes-Merton (BSM) model provided the first mathematically sound formula to price European style option. In this model, the dynamics of the underlying stock return take the form $dS_t / S_t = rdt + \sigma dW_t$, where the drift r is the risk-free rate of return, and the volatility σ is assumed to be a constant. This is equivalent to say that the stock price process is a Geometric Brownian Motion and implies that the stock return at time t is normally distributed with mean rt and variance $\sigma^2 t$.

Under these assumptions, one can derive explicit formulas to price European options written on stock S_t with strike K and expiry T . The price of a European call option at time t is $C(S_t, t) = N(d_1)S_t - N(d_2)Ke^{-r(T-t)}$, while the price of a European put option at time t is $P(S_t, t) = Ke^{-r(T-t)}N(d_2) - S_t N(-d_1)$, where N() is the cumulative density function for a standard normal distribution, and,

$$d_1 = \ln(S_t / K) + (r + \sigma^2 / 2)(T - t) \Big/ \sigma\sqrt{T - t}$$

$$d_2 = \ln(S_t / K) + (r - \sigma^2 / 2)(T - t) \Big/ \sigma\sqrt{T - t}$$

Notice that when we choose certain strike price and expiry, the Black-Scholes-Merton formula becomes a one-to-one correspondence between option price (C or P) and volatility σ: option price= $f_{BSM}(S_t, t, K, T; \sigma)$. The option price will be higher if the underlying stock price (or return) is more variable (i.e. has higher volatility σ).

1.4 Implied volatility

Given the observed European option price V^{obs} for a contract with strike price K and expiry T, the implied volatility IV is defined to be the value of the volatility parameter that must go in to the BSM formula to match this observed price:

$V^{obs} = f_{BSM}(S_t, t, K, T; IV)$

If the BSM model is accurate, we must have $IV(t, S_t, K, T) = \sigma$ for any K and T. But this does not hold for real-life market prices.

1.5 Monte Carlo option pricing

The Monte Carlo estimator of option price is the average of discounted payoff under the arbitrage-free martingale measure. The algorithm looks like:

Step1: *Get N random samples of* S_T , $\{S_T^{(1)}, S_T^{(2)}, ..., S_T^{(N)}\}$ *(If* S_T *follows certain distribution, generate directly from this distribution; If not, simulate N trajectories of the process under martingale measure to get the end price* $S_T^{(i)}$ *)*

Step2: *Calculate the payoff* $V_T^{(i)} = payoff(S_T^{(i)})$ *for each i.*

Step3: *The option price at* time t=0 is estimated to be $\frac{1}{N}\sum_{i=0}^{N} e^{-rT} V_T^{(i)}$, where e^{-rT} is the discount factor.

The Law of Large Numbers ensures that this estimation converge to f(X) when

$N \to \infty$, while the Central Limit Theory shows that the error of this estimation is $O(1/\sqrt{N})$.

1.6 Numerical solution of SDE's (Euler scheme)

To simulate the trajectory of a random process X, $dX_t = f(t, X_t)dt + g(t, X_t)dWt$, over the time interval [0,T], one simple way is to use the Euler scheme. To do this, we discretize the interval in to N subintervals and denote T/N by Δt, $t_i = i\Delta t$,i=0,1,…N.

To estimate the value of X at time t_i , based on the information from the previous time step t_{i-1} , the Euler scheme provides the following formula:

$X_{t_i} = X_{t_{i-1}} + f(t_{i-1}, X_{t_{i-1}})\Delta t + g(t_{i-1}, X_{t_{i-1}})(W_{t_i} - W_{t_{i-1}})$ .We assume that the value of X at time 0 is X_0.

Euler scheme is strongly convergent with order 1/2, which indicates that $E|X_\delta(T) - X(T)| \leq C\delta^{1/2}, \forall \delta < \delta_0, \forall T$. Here X_δ is the time-discretized approximation of the continuous-time process X, with δ as the maximum time increment of the discretization.

1.7 Stochastic volatility model

Instead of assuming the volatility to be a constant, a more sophisticated but realistic approach is to treat volatility as a random process V_t . Commonly, we choose V_t to be a mean-reverting process, like the Ornstein–Uhlenbeck (OU) process, the dynamic of which takes the form $dV_t = \alpha(m - V_t)dt + \beta dW_t$. Here α represent the speed of reverting, while m and β represent the mean and volatility of the OU process respectively.

Under the risk-neutral martingale measure, the stochastic volatility model looks

like:

$$dS_t = rdt + \sigma_t dW_t^{(1)}$$

$$\sigma_t = |V_t|$$

$$dV_t = \alpha(m - V_t + g(\gamma, \rho))dt + \beta dW_t^{(2)}$$

Note that there is an extra term appears in the drift of OU process to satisfy the no-arbitrage condition of the market. Here γ represents the market price of volatility and ρ represents the correlation between the two Wiener process $W_t^{(1)}$ and $W_t^{(2)}$ that drive the stock process and respectively. And $g(\gamma, \rho)$ can be written as $\rho\frac{\mu - r}{\sigma_t} + \gamma\sqrt{1-\rho^2}$.

2. Statement of the problem and its main challenges

2.1 Deviation of the BSM model from the reality

As said earlier in chapter 1.4, the BSM's assumption about constant volatility does not hold in general. Instead, we can observe a pattern called "volatility smile" from real option market. Keeping parameters S_t, t and T fixed, solve $V^{obs} = f_{BSM}(S_t, t, K, T; IV)$ for the implied volatility IV for pairs of corresponding strike price K and observed option price V^{obs} values. We get a function of IV with respect to K (or K/S_t). The graph of this function, also called as "volatility smile curve", is typically downward sloping when K is smaller than or near to the current stock price S_t (moneyness$\leq$1), while it is upward sloping for K greater than S_t (moneyness>1) as shown in figure 2.1.

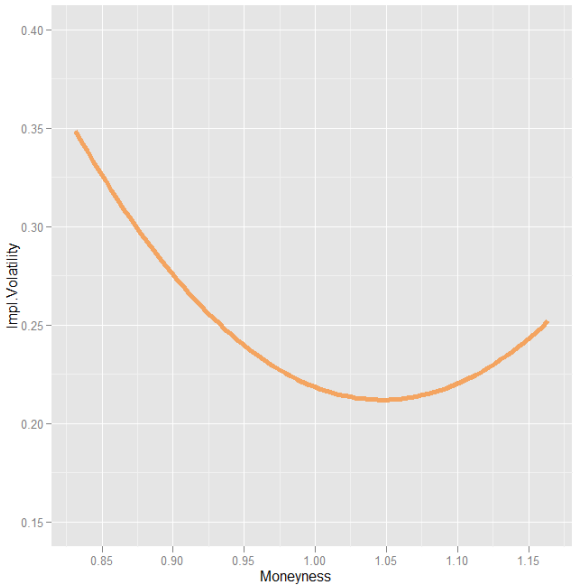

Figure 2.1 Volatility smile curve

The BSM model also assumes that the underlying stock returns follow Normal distribution (mentioned in chapter 1.3). This assumption is often violated in practice. The common opinion in finance is that Normal model tends to under-evaluate investment risk – extreme cases are more likely to happen than that predicted by Normal distribution. Tremendous losses, such as market crashes, would happen with a probability close to 0 in a "Normally distributed world". But in reality, we experienced market crashes approximately every 10 years. Hence, the real returns of stocks should have heavier tails than the normal distribution as shown in figure 2.2, which allocate more probabilities to large losses and large gains.

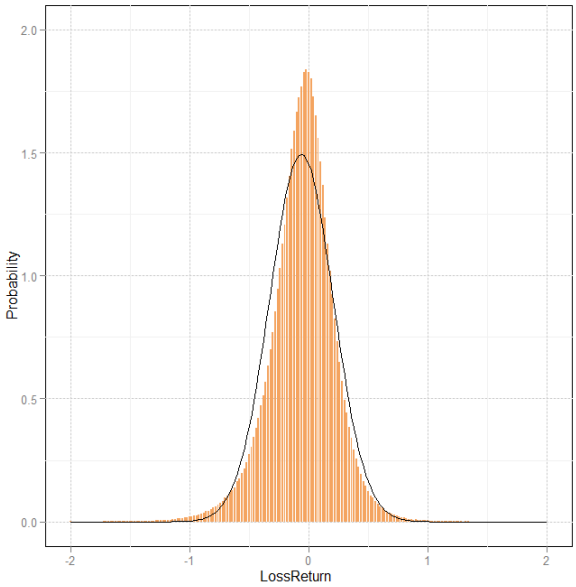

Figure 2.2 Real stock return (histogram) compared to normally distributed return (black curve)

2.2 Goal and purpose

Volatility smile is something we can observe from the option market and heavier tails of return distributions are what we expect in stock market – these two should be connected in the real world. The major goal of this project to is to find the heavy-tailed distribution that yields the correct smile curve as observed from the option market data.

Once we have determined the real return distributions, we can extract useful information about the underlying stocks form it. Risk measures, such as Value-at-Risk (VaR) and Expected Shortfall (ES) can be calculated directly from those distributions. Notice that the future return distributions determined this way contain the perceptions of option market participants. As a result, information from such distributions can be very helpful for investors to better manage the investment risk.

2.3 Challenges

The real return distributions are very difficult to determine statistically. First of all,

they belong to no known parametric families. The only information we know is that the distributions have heavy tails. Second, since the tails represent rare events, to correctly estimate them, extremely large number of outcomes are needed. For example, if the market crashes happened once every ten years, to correctly determine the probability of crashes statistically, one needed stock market data for more than 100 years, which is impossible. Also, it's inappropriate to use data from a long time ago to estimate the future distribution.

3. Proposed approach

The idea of the proposed approach is to get return distributions by fitting suitable model to the option market data. The "suitable model" here refers to the stochastic volatility model mentioned in chapter 1.7, which provide the dynamics for both the stock process and volatility process:

$$dS_t = rdt + \sigma_t dW_t^{(1)}, \quad \sigma_t = |V_t|$$

$$dV_t = \alpha(m - V_t + g(\gamma, \rho))dt + \beta dW_t^{(2)}$$

How can we choose those parameters appear in the SDEs ? The answer is to use the "inverse approach": we first build the model with the parameters as unknown variables, and then keep changing the parameters iteratively, implement the model and get several outcomes based on various sets of parameters. In this process, there will be one special set of parameters which brings the outcome closest to our target outcome. This is the set of parameters we want to use for the model.

For the inverse problem in this project, the "outcomes" are the smile curves computed from the stochastic volatility model and the "target outcome" is the smile curve observed from option market data. The detailed steps can be explained by the following flow chart:

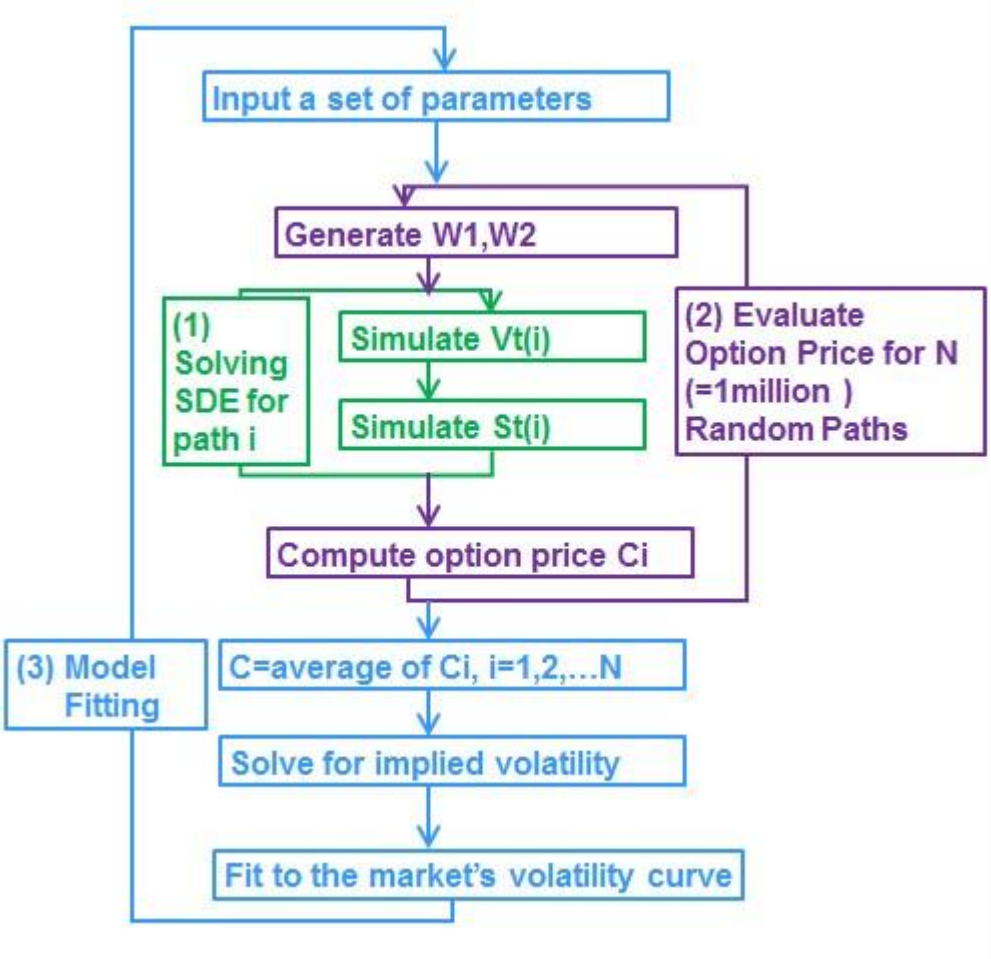

Figure 3.1 Flow chart of the model fitting approach

The outer-most loop in the flow chart represents the model-fitting process. Once we input a set of parameters, we will get a smile curve from simulation and compare it to the market smile curve. To determine how close the two curves are, we construct an objective function $f(\vec{x})$, here $\vec{x} = (\gamma, \rho, \alpha, m, \beta)'$. If $\{a_1, a_2, ...a_p\}$ are the p points that form the market smile curve and $\{b_1(\vec{x}), b_2(\vec{x}), ..., b_p(\vec{x})\}$ are the corresponding p points that form the simulated smile curve using parameter vector $\vec{x}$, then the value of the objective function is $f(\vec{x}) = \sum_{i=1}^{p} w_i (a_i - b_i(\vec{x}))^2$, where the w_i are some constants representing the weights we want to give to the various points. We fit the model to the market data by finding the "optimal" parameter vector $\vec{x}$ that minimizes the objective

function f. Computationally, we find the optimal parameter vector using the Nelder-Mead multi-dimensional iterative optimization algorithm implemented in the R computational statistics environment. The advantage of the Nelder-Mead method is that it is robust and uses only function value of $f(\vec{x})$, requires no derivatives. However, the convergence is relatively slow; it often requires 800-1200 iterations to find the optimum.

The middle loop of the flowchart implements Monte Carlo option pricing (with sample size N) as described in chapter 1.5. Notice that once we change the volatility term in the SDE of the stock process from a constant to a random process, we no longer know the distribution of the stock price at expiration date. The only way to get $S_T^{(i)}, i = 1, 2, \ldots N$ is to simulate the whole trajectory of the stock process. That is why we have the inner-most loop, which represents the simulations of the trajectories for both the volatility process and the stock process. The simulation is done by the Euler scheme mentioned in chapter 1.6.

To sum up, in each of the 800-1200 optimization iterations, Monte Carlo method needs to be done with sample size N, and each of the N samples requires the simulation of M points on the trajectories for volatility and stock processes. Overall, the whole inverse approach is extremely compute intensive, mainly due to the fact that N need to be as large as 1 million to accurately reflect the tails of the return distribution.

4. Use of massively parallel computing

4.1 The need for high performance parallel computing

In chapter 3, we introduced the model-fitting approach. Once we get the set of optimized parameters, we can use them to simulate the stock price process and then get the future return distribution. Note that those parameters and corresponding distribution vary among different stocks, and they are also

changing over time. If we want to manage the risk of a portfolio that contains several stocks in a timely manner, the whole model-fitting process (as described in the flowchart) is needed to be redone many times. Thus, it is curial that a single model-fitting process (for one stock and one day) can be done within a reasonably short time.

If we implement our model on the CPU, computing the steps in the three layers of loops sequentially, the time to find the set of optimized parameters for one stock and one day would be 75 hours (about three days). Obviously, this approach is not feasible because we may miss the best opportunity to adjust the portfolio and the anticipated future return distribution may have already changed during these 3 days.

The reason why this process is so time-consuming is that the sample size N of the Monte Carlo simulation needs to be as large as 1 million. But one good thing is that the sample paths are independent of each other, thus, the Monte Carlo option pricing loop in the model-fitting process can be parallelized. That is, instead of simulating N stock price paths one after another, we want to simulate them simultaneously. To do this, we introduce massively parallel GPU computing in to our project.

4.2 GPU computing and CUDA

A GPU (graphic processing unit) has far more processing cores than a common CPU has. With a limited number of sophisticated ALUs (algorithm and logic units), a CPU is most suitable to perform fast sequential operations. In contrast, GPU is designed to process multiple pixels at one time and has hundreds of parallel cores, which is ideal for data-parallel operations. In our case, the Monte Carlo path generations are exactly “data-parallel operations”: the generation of each path is using same commands (Euler scheme) but must be executed on different data (normally distributed increments).

Controlling the GPU requires specialized software tools and language. We use

Nvidia's CUDA (Compute Unified Device Architecture), which is an extension to the standard C language, and some other tools like nvcc CUDA –C compiler, CUDA runtime and mathematical libraries. Briefly speaking, CUDA provided a way that one could compile some kinds of special programs in Visual Studio, which will be executed on GPU. This is done by "kernel calls". A "kernel" in CUDA refers to a special type of function which will be running on GPU but can be invoked by CPU code. Once a kernel function is called, a grid of parallel threads will be generated by CUDA, and they are further equally divided into blocks. One block of threads will be assigned to the same stream multi-processor and execute the same commands. In our case, each thread is responsible for the simulation of one volatility trajectory and one stock trajectory. Thus, our kernel function needs to generate 1 million parallel threads.

4.3 Performance Optimization

To achieve best performance on GPU, several steps of optimization are needed to be done.

First of all, one needs to choose appropriate size for blocks. The goal is that there must be an overwhelming number of blocks to saturate GPU processors with jobs waiting for execution. Yet, choosing the number of blocks to be the maximum 1 million is still not good enough. Previously, we introduced that block is the unit for thread organization. But in terms of thread execution, the unit is a "warp" and each warp contains 32 threads. Thus, the dimension of one block is best to be the multiple of 32. In our project, we choose the number of threads in one block to be 32 and there will be 31250 of such blocks in total.

Secondly, we need to make good the use of high speed on-ship memory, also known as shared memory. In our Monte Carlo path generation, the first step is to generate the normal distributed increments (dW) for both the volatility and stock processes. Altogether, there will be 2*1million*256 such increments (of float type) generated and stored in GPU global memory. For each thread to complete

the simulation of an entire volatility path and an entire stock process, it has to visit the global memory for 2*256 times to get the increments, which is a great waste of time. One approach is to upload those random increments into high speed shared memory. It is much faster for threads to access on chip shared memory than global memory. However, for each block, the size of shared memory is limited to be 16384 bytes (4096 floats), it is impossible for each thread to upload all the increments it need to complete the whole path simulation at a time, given that there are 32 threads in each block. The solution is to divide the path generation (containing 256 time points) into 8 stages. At the beginning of each stage, we upload the 2*32*32 random increments need for this stage in to the pre-allocated shared cache, and then begin the simulations using the end values from previous stage as the initial values. To get around the limited resources of shared memory, replacements are done in place – that is, the random increments in shared cache are gradually replaced by the simulated trajectories. After each stage, the end value of stock process and volatility process are stored and the shared cache is then filled with random increments for the next stage.

Finally, we need to coalesce global memory access to reduce effects of high memory latency. Although we have made good use of high speed shared memory, we still need to access to global memory 8 times to simulate the entire trajectories. To speed up this process, the best thing we can do is to let the 32 threads in a block accessing the 32 continuous addresses in the global memory – this is known as the memory coalescing technique.

After these improvements, to finish the whole model-fitting process for one stock and one day, it only takes 15 minutes when we implement the Monte Carlo path generation part into parallel GPU computing. There’s an approximately 300 times speed-up compared to the one done purely by CPU.

5. Use of penalty functions (soft constraints) and data smoothing

5.1 The structure of output files

The high performance GPU computing introduced in Chapter 4 made it possible to implement the model-fitting process for several stocks each day in real time. Below is a snap shot of the one of the output file:

Ticker	MSFT				
Expiry	8/17/2007				
Current Date	7/6/2007	7/9/2007	7/10/2007	7/11/2007	…
Input					
DTM	30	29	28	27	…
Interest Rate	0.0118306	0.027561	0.0317223	0.03949	…
S0	29.97	29.87	29.33	29.49	…
Strike prices	(vector)				…
Market IV	(vector)				…
Output					
Optimized IV	(vector)				…
Objective Value	1.31E-05	3.65E-05	7.58E-05	5.06E-05	…
gamma	0.2638264	0.239423	0.0640914	0.262009	…
rho	-0.149856	-0.15625	-0.165675	-0.18285	…
alpha	2.6915851	2.463546	2.3344001	2.297636	…
m	0.0211545	0.063173	0.0772335	0.098175	…
beta	0.4719973	0.438384	0.3945709	0.390613	…

Table 5.1

This is an example of the contract MSFT-08/17/2007. Each column represents the result from fitting the model for one stock (MSFT) and for one day (Current Date). Begin with days to maturity (DTM) =30 and moving towards the expiry, we observed different stock prices (S0), market implied volatility (Market IV) each day and use them as input. Then, in the second part of the table, we get the optimized parameters (gamma,rho,alpha,m,beta) and their corresponding simulated implied volatilities (Optimized IV) for each data date.

To get reasonable optimized parameters and to further obtain stable risk measures (which will be explained in Chapter 6) for each date, we need to

employ two special techniques: using penalty functions and smoothing inputs.

5.2 Penalty functions

Previously in Chapter3, we have constructed an objective function, f (gamma, rho, alpha, m, beta), and we said that the way to found the optimized set of parameters is to plug f into the Nelder-Mead optimizer. The Nelder-Mead optimizer is stable for non-differentiable objective functions, like our f( ), but one drawback is that this method does not accept constraints on parameters. That is, for each parameter, it is possible for it to go from negative infinity to infinity, which is unwanted.

In our case, some of the parameters must be bounded to ensure reasonable simulations. Firstly, the parameter “m” represents the mean value of the volatility process, which should be positive. Secondly, the parameter “rho” represents the correlation between stock price and the level of volatility in the market, thus must be negative. Finally, the parameter “gamma” represents the market price of stochastic volatilities. Although it is hard to say the “gamma” should be bounded within some specific values, intuitively, “gamma” cannot be too large and we can conclude from past experience that large “gamma” will cause troubles thus it is unwanted.

Although sometimes we could get simulated smile curve very close to the market one based on negative “m”, positive “ rho” or very large “gamma”, those optimized parameters go against their financial meanings, and will cause troubles later on when we use them to measure the risk.

Since we are unable to add hard constraints like “m>0” to the optimizer, one way to bounded the optimized parameters is to use penalty function, also known as a “soft constraint”. To do this, we plug into the optimizer a new function g(gamma, rho, m, alpha,beta), which is constructed by :

g(gamma, rho, alpha, m, beta)

=f(gamma, rho, alpha, m, beta)+P(gamma, rho, m).

Here P() is the penalty function, which is a weighted sum of the three individual penalties for gamma, rho and m. Since we want to minimized the value of g(), once a parameter goes beyond its proposed boundary, the P() function will take on positive values, which slow down the minimizing of g(). By doing this, large “gamma”, positive “rho” and negative “m” will be punished during the optimization process. In addition, care is taken to assure that the distance f and penalty P components of the penalized objective are of comparable size.

The soft constraint technique made it possible for us to obtain reasonable optimized parameters. Notice that in the table shown at the beginning of Chapter 5, the numbers in row “gamma” are all reasonably small and we have all negative numbers in row “rho”, positive numbers in row “m”.

5.3 Smoothing input data

Not all the market-observed implied volatilities (or option prices) can reflect the real market condition accurately. Say, at a specific day, the trading volume of a MSFT option with certain strike price can be very small, which results in odd value of implied volatility. To diminish that kind of effect, we are using the

technique called input smoothing. The idea is simple: instead of inputting the market implied volatilities for one day into the optimization process, we input the average of a week's implied volatilities. For example, the "mktIV" at DTM=30 in the table is actually the average of the market implied volatility from five days: DTM=30,31,32,33,34; and the "mktIV" at DTM=29 is the average from DTM=29,30,31,32,33; so on and so force.

If intra-day option and stock prices are available, then the averaging can take place over the implied volatilities observed during the same day.

6. Results and findings: Measures of risk

6.1 Return distribution and risk measures

After we found the optimized parameters, which bring the simulated smile curve closest to the market smile curve, we can use them to form the stock return distribution. For the convenience of comparison, we want the return distribution for all stocks and all dates to be annualized and risk-neutral. To do this, for each stock and each date, plug the corresponding optimized parameters and T=1 into the path-simulation algorithm (the inner-most loop in flowchart), and get the stock prices at the final points of the 1 million sample paths, $S_T = \{S_T^{(1)}, S_T^{(2)}, \ldots S_T^{(N)}\}$. Then, to convert the year end prices into risk-neutral return, we are using : $R(i) = \log(S_T^{(i)} / S0) - r$. The 1 million outcomes R={R(1),R(2),…R(N)} contain all the information we want about the return distribution.

As a continuation of table 5.1, below are the risk measures for ticker MSFT based on the information from contract 07/18/2007:

Ticker	MSFT				
Expiry	7/18/2007				
Current Date	7/6/2007	7/9/2007	7/10/2007	7/11/2007	…
DTM	30	29	28	27	…
Output: risk measures					
VaR 5%	-0.47593	-0.49369	-0.49713	-0.50228	…
VaR 1%	-0.72548	-0.75393	-0.75822	-0.76687	…
VaR 0.1%	-1.08199	-1.13121	-1.13484	-1.14714	…
VaR 0.01%	-1.47181	-1.52259	-1.53554	-1.53317	…
VaR 95%	0.257355	0.266021	0.267935	0.259498	…
VaR 99%	0.435004	0.449728	0.452899	0.440269	…
VaR 99.9%	0.67957	0.703477	0.710497	0.694464	…
VaR 99.99%	0.93372	0.965724	0.966211	0.953331	…
ES 5%	-0.63122	-0.65648	-0.66026	-0.66686	…
ES 1%	-0.88091	-0.9191	-0.9237	-0.93223	…
ES 0.1%	-1.24483	-1.29944	-1.31007	-1.31665	…
ES 0.01%	-1.63603	-1.70682	-1.7326	-1.74169	…
ES 95%	0.367442	0.380412	0.382752	0.37227	…
ES 99%	0.542443	0.561905	0.565338	0.551676	…
ES99.9%	0.791389	0.820218	0.825364	0.806448	…
ES 99.99%	1.024343	1.076397	1.083471	1.057516	…
Mean	-0.08654	-0.08873	-0.08842	-0.09405	…
Standard Deviation	0.227276	0.235459	0.236841	0.236054	…
Skewness	-0.44625	-0.47165	-0.47877	-0.50717	…
Kurtosis	4.662717	4.743076	4.754733	4.784459	…

Table 6.1

At the bottom part of the chart, we have the four most important moments of the return distribution: mean standard deviation, skewness and kurtosis. Skewness is defined as $\frac{E[(R-mean)^3]}{(E[(R-mean)^2])^{3/2}}$ to measure the asymmetry of the distribution, while kurtosis is defined as $\frac{E[(R-mean)^4]}{(E[(R-mean)^2])^2}$ to measure the "peakedness" of the distribution. If returns were normally distributed, they would have skewness=0 and kurtosis=3. Here we have negative skewness and larger kurtosis than that of normal distribution. Negative skewness indicates that the tail on the left side of the probability density function is longer than the right side. For return distribution, it means that extreme losses are more likely to

happen than extreme gains. Higher kurtosis indicates that more of the variance is the result of infrequent extreme deviations, as opposed to frequent modestly sized deviations. That is, if two distributions have the same variance, the one with higher kurtosis will have heavier tails. For return distribution, it means that the probabilities for large losses and gains are higher than normal distributed returns.

"VaR "stands for" Value at Risk, which is the most widely used numerical measure of risk. Briefly speaking, VaR p% is the p% quantile of the return distribution. VaR 5%,1%,0.1% and 0.01% contains the information about losses, while VaR 95%,99%,99.9% and 99.99% contains the information about gains. For instance, VaR 1%=-0.72548 indicates that the probability for the annual rate of return less than -0.72548 is 1%, that is, with 99% confidence, we know that the stock price after 1 year will not fall below S0*exp(-0.72548)=0.48*S0. Likewise, VaR99%=0.435004 indicate that we know with 99% confidence that the stock price after 1 year cannot rise above S0*exp(0.435004)=1.54.

"ES" stands for "Expected Shortfall", which is another important risk measure. It is also known as the conditional Value at Risk or average Value at Risk. For example, $ES0.1\% = mean(R(i) \mid R(i) < VaR0.1\%)$.

6.2 Evolution of return distributions and risks over time

In table 6.1, each column represents a prediction of the future stock return. The difference is that they are based on different information: for example, stock return distribution in column DTM=30 reflects the market participants' perception on 7/6/2007; when it comes to 7/9/2007, those participants may change their perception based on some news they heard about MSFT or any other kind of information during the three days, which brings a slightly different version of future return distribution in column DTM=29.

Once we've got the output file for a stock like table 6.1, we can easily observe the evolution of return distribution and risks over time. For the ticker MSFT, the

return distribution basically remains the same from DTM=30 (7/6/2007) to the expiry (7/18/2007). Below are how the return density look like on DTM=30, DTM=20 and DTM=10 respectively.

Evolution of implied return distributions over time: MSFT

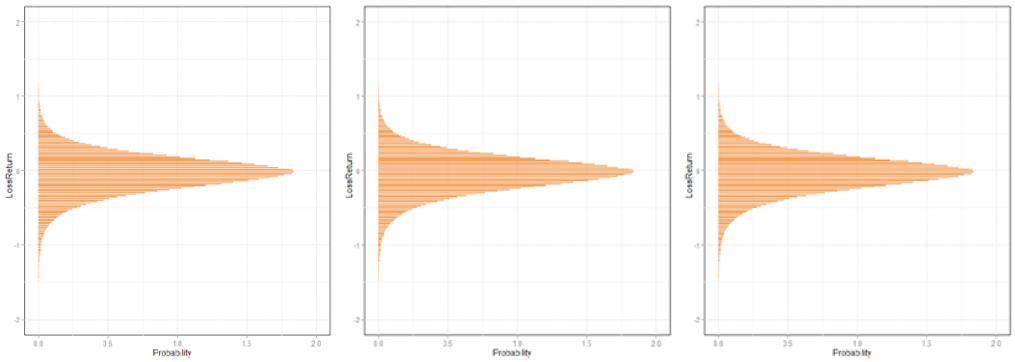

Figure 6.1
MSFT DTM=30

Figure 6.2
MSFT DTM=20

Figure 6.3
MSFT DTM=10

Different from MSFT, if we take a look at the output file for ticker CSCO with the same expiry 7/18/2007, we will find that the market perspective for CSCO stock return changed significantly during the same month. Figure 6.4, 6.5 and 6.6 are the return distribution densities at DTM=30, DTM=20 and DTM=10 respectively. As time approaches expiry, the return distribution from market participants' perception becomes more and more "spread out", which indicates that people somehow felt that the potential risk to invest in CSCO stock was growing.

Evolution of implied return distributions over time: CSCO

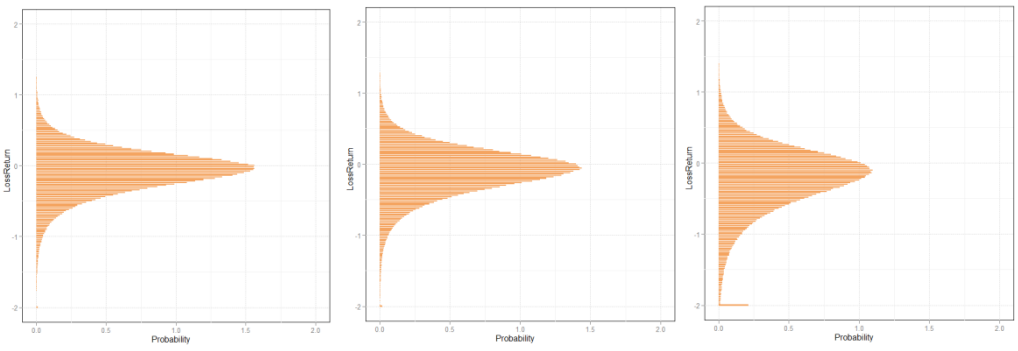

Figure 6.4
CSCO DTM=30

Figure 6.5
CSCO DTM=20

Figure 6.6
CSCO DTM=10

The evolution of VaR values for CSCO and MSFT in figure 6.7 tells the same story:

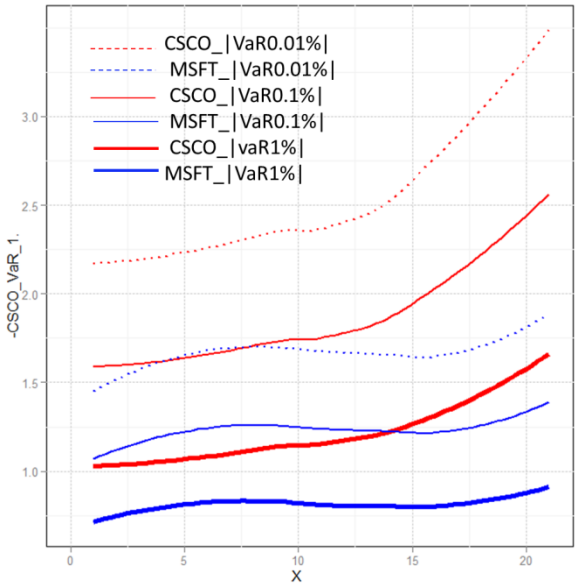

Figure 6.7 Evolution of Value-at-Risk (VaR)

In figure 6.7, the absolute value of VaRs from loss side (0.01%,0.1%,1%) are plotted for ticker MSFT and CSCO respectively. In the course of one month, while the VaRs for MSFT extracted from option data did not changed so much, those for CSCO had increased sharply. Yet, the stock price of Cisco remained virtually unchanged during the same period. This means that the information contained in figure 6.8 could not have been obtained by observing only the stock market.

The information about the evolution or risks could help investors to adjust their strategy in time. For example, based on the information showed in figure 6.7, an investor who originally included certain shares of CSCO in his portfolio may consider reducing his exposures to CSCO to keep the risk of his portfolio in a low level.

6.3 Comparison of return distributions across different stocks

The flexibility of our modeling approach made it possible for us to fit our stochastic volatility model to a wide variety of companies' stocks. And the high performance GPU computing made it possible for us to get the output files like table 6.1 for several stocks each day. At a given day, we can compare the return distributions across different stocks.

Again, we take an example of the contracts expired on 8/17/2007 and we consider the ticker MSFT, INTC and F. Standing at 7/20/2007 (DTM=20), we got their return distribution densities shown in figure 6.8,6.9 and 6.10. Those three densities illustrate the wide variation of the market participant anticipation about the future returns expected from different companies.

Return distribution of the different stocks on the same day

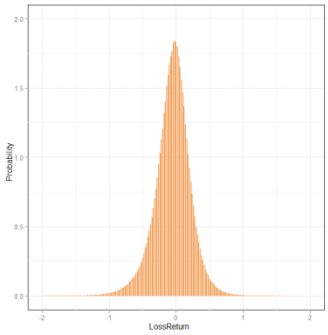

Figure 6.8
MSFT DTM=20

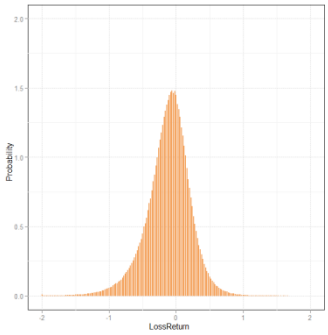

Figure 6.9
INTC DTM=20

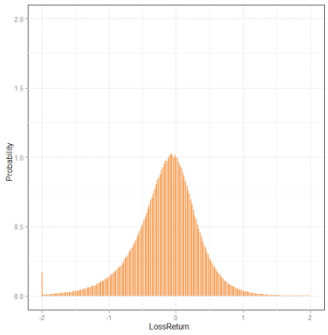

Figure 6.10
F DTM=20

A comparison of risk measures across these three stocks is also available:

	MSFT	**INTC**	**F**
VaR 5%	-0.5138	-0.6973	-1.0095
VaR 1%	-0.8177	-1.0772	-1.5924
VaR 0.1%	-1.2653	-1.6259	-2.4342
VaR 0.01%	-1.7291	-2.1727	-3.2992
VaR 95%	0.3441	0.3604	0.5672
VaR 99%	0.5749	0.6138	0.9624
VaR 99.9%	0.906	0.9679	1.5177
VaR 99.99%	1.2622	1.3326	2.0999
Mean	-0.0621	-0.1263	-0.1577
Standard Deviation	0.267	0.3282	0.49

Table 6.2

Table 6.2, figure 6.8, 6.9 and 6.10 shows us the same story: If we rank these three stocks in terms of risk, it will be: MSFT<INTC<F. Investors can make good use of this information to form their portfolio. For a risk-averse investor, it is better for him to allocate more exposure on stocks like MSFT; while for a risk-seeker who would like to take on higher risks for large gains, he may want to choose stocks like Ford.

7. Comparison between different models

7.1 Introduction to Heston model

Previously in chapter 1.7, we introduce the stochastic model based on Ornstein–Uhlenbeck (OU) process. Another possible stochastic model is the Heston model, which takes the form:

$$dS_t = rdt + \sigma_t dW_t^{(1)}, \quad \sigma_t = \sqrt{V_t}$$

$$dV_t = \alpha(m - V_t + g(\gamma, \rho))dt + \sqrt{V_t}\beta dW_t^{(2)}$$

Different from the OU model, the process Vt here represents the variance process and the volatility term is the square root of Vt. Using the same model fitting approach as described in chapter 3, we are able to fit the Heston model to the option market data and extract risk measures.

7.2 Comparison between Heston model and OU model

For some data dates, OU model gives us a better fit to the market smile curve; while for some other dates, Heston model may work better. But for most of the cases, like the MSFT 8/17/2007 contract, the two models gave approximately the same accuracies of fits.

Take an example of the case DTM=20 for this contract, the market implied volatility curve is available for moneyness from 90% to 110%. The far out-of-the-money or far in-the-money options had very low liquidity, thus we could not calculate reliable implied volatilities based on their prices – this is true for most of the data dates. The simulated smile curves under both the OU model and Heston model fit to the market smile curve very well for moneyness from 90% to 110%.

However, when we extend the two simulated smile curves for moneyness from 50% to 150%, we get the following graph:

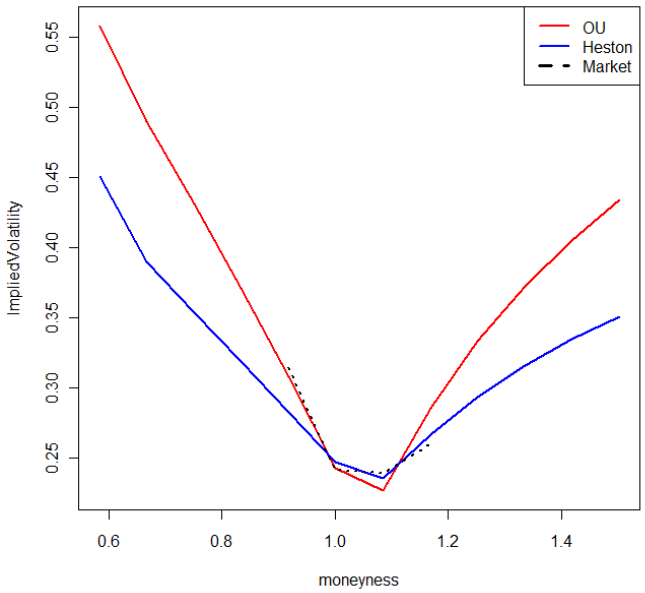

Figure 7.1 Volatility smiles based on different models

The farther we get in-the-money or out-of-the-money, the more the two curve diverge. The smile curves under OU model tend to be steeper and more pronounced than that under Heston model, with higher implied volatilities at both sides. Corresponding to figure 7.1, risk measures extracted form market data based on these two models are also different:

	OU	Heston
VaR 5%	-0.513821386	-0.370221775
VaR 1%	-0.817692315	-0.564807189
VaR 0.1%	-1.265257085	-0.831924802
VaR 0.01%	-1.729066825	-1.080225329
VaR 95%	0.344071326	0.266760695
VaR 99%	0.57491967	0.420942406
VaR 99.9%	0.906041173	0.627273124
VaR 99.99%	1.26223359	0.817352383
Mean	-0.062128472	-0.038902367
Standard Deviation	0.26695421	0.196178877
Skewness	-0.410082946	-0.275810662
Kurtosis	5.117631168	4.105952588

Table 7.1

Both table 7.1 and figure 7.1 told us the same story: Compared to OU model previously used in Chapter 6, risks tend to be “under-estimated” in Heston model. This is not only true for ticker MSFT at 7/20/2007, in fact, it’s a common

phenomenon for most of the data date when both the OU model and Heston model could fit to the market.

7.3 Further discussion

Although it is hard to say which model is better in general, our flexible model-fitting approach implemented by high performance GPU computing enable us to compare different stochastic models. In the area of investment risk management, “model risk” plays an important role. Given the information like table 7.1, risk managers can make better decisions to choose appropriate model for their specific problems.

8. References

[1]Jean-Pierre Fouque, George Papanicolaou and K.Ronnie Sircar, “Derivative in Financial Markets with Stochastic Volatility”, Cambridge University Press,2000.

[2]Jim Gatheral, “The Volatility Surface”, Wiley Finance,2004.

[3]Stefano M. Iacus, “Simulation and Inference for Stochastic Differential Equations”, Springer, 2008.

[4]Thomas Mikosch, “Elementary Stochastic Calculus”, World Scientific, 1998.

[5]Paul Glaseerman, “Monte Carlo Method in Financial Engineering”, Springer, 2004.

[6]Jason Sanders and Edward Kandrot, “CUDA by Examples”, Nvidia, 2010.

[7]David B. Kirk and Wen-mei W. Hwu, “Programming Massively Parallel Processors”, Nvidia, 2010.

[8]WPI research report “Restructuring Option Chain Data Sets Using Matlab” written by Alison Wooden, May 2010.

[9] WPI research report “Financial Option Trading Data Analysis” written by Jun Zhang, Dec 2010.