Spring 5-2016

Stochastic Modeling of US Equity Returns

Mitchell Benoit
University of Maine

STOCHASTIC MODELING OF US EQUITY RETURNS

By

Mitchell Benoit

A Thesis Submitted in Partial Fulfillment
of the Requirements for a Degree with Honors
(Physics)

The Honors College
University of Maine
May 2016

Advisory Committee:
Dean Astumian, Professor of Physics, Advisor
Pankaj Agrrawal, Associate Professor of Finance
David Batuski, Professor of Physics
Charles Hess, Professor of Physics
David Gross, Adjunct Associate Professor in Honors

Abstract

A non-linear Langevin equation was constructed in order to approximate the behavior of the S&P 500 Index intraday price movements. Price changes were assumed to be a function of supply/demand offsets that resulted from new information. Based on this observation, I constructed an equation that describe the rate of return for the market data, which depended on (1) the sensitivity of the market to supply/demand offsets, (2) the liquidity of the market, (3) the memory effects of recent returns, and (4) the memory effects of the volatility of recent returns. MATLAB was used to find appropriate coefficients for these terms in the Langevin equation. I found that the Langevin model was capable of very accurately modeling the volatility and liquidity of the market, however, it was more difficult to model memory effects. Overall, I found that data sets with larger geometric mean values had a commensurately larger alpha and beta value, which suggests that the returns experienced in the short term are due to the dependence of future market prices on past market prices. This model approximates market returns over short periods of time—approximately one day—and should not be expected to remain valid over long time periods, given that underlying market conditions that are assumed constant in this Langevin model would be expected to change.

Acknowledgements

I would like to thank Professor Astumian for his continued support and encouragement as I have worked on my Honors Thesis. His experience in biophysics has given him a rigorous perspective on stochastic processes, and he has encouraged me to question assumptions when modeling financial systems. I would also like to thank Professor Agrrawal for his perspective on the financial markets, and his valuable insights regarding quantitative financial modeling. I am appreciative that the University of Maine has provided all undergraduate students with access to MATLAB. Thank you as well to my friends and family who have supported me throughout this thesis.

Table of Contents

List of Tables

List of Figures

I. Brownian Motion Background

Brownian motion was first observed by the Scottish Botanist Robert Brown when viewing a small particle suspended in a fluid under a microscope. The particle exhibited random motion as if it were alive. In order to ensure living organisms within the fluid were not causing this motion, later tests were performed on samples that were taken from water trapped for years in ancient rocks. Even this "dead" water resulted in suspended particles jittering. It has since been determined that the random motion of the particle is caused by the uneven forces of the water molecules colliding with the small particle. On average, the net force of these collisions is zero. However, due to the small size of the particles, it is unevenly struck by the water atoms, and therefore it seems to move around. Over the long term, since the net force is zero, the average displacement is expected to be zero as well.

In 1905, Einstein published "Investigations on the Theory of the Brownian Movement" in which he determined the root mean squared displacement of a Brownian particle. This analysis was also useful in demonstrating the size of atoms [3].

(1) $$p(x,t) = \frac{N}{\sqrt{4\pi Dt}} e^{-\frac{x^2}{4Dt}}$$

(2) $$\overline{x^2} = 2Dt$$

Equation one describes the probability distribution of N particles subject to a diffusion coefficient D. In general, an individual Brownian particle's movement is not a Markov process due the the effects of the inertia of the particle. When the particle is in motion, it tends to continue in that direction, and therefore each time step is not uncorrelated with past locations of the particle. Under appropriate circumstances where the viscous drag term is sufficiently large, the inertial component can be neglected [4].

Often, a Langevin equation is used in physics to describe the motion of a Brownian particle. For simplicity, when deriving the a Langevin equation we will assume a continuous Markov process. We are interested in the movement of a particle as a function of the forces that act on it. Neglecting the noise term, we can conclude that the rate of change of the position is due to the force. The particle exists in a field that acts on it with a force A, which is a function of its position and time.

(3) $$\frac{dX(t)}{dt} = A(X(t), t)$$

This equation can be modeled numerically as an "update" equation, which will have the form:

(4) $$X(t + dt) = X(t) + A(X(t), t)dt$$

In Equation 4, we see that the future step $X(t + dt)$ is a result of the increment Adt added to the previous value $X(t)$. This technique is called Euler's method.

We would also like to incorporate a random variable into this equation, which will account for the random collision of molecules with the particle. Specifically, we will model the noise term as Gaussian whose density function is given by:

(5) $$P(x) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left(-\frac{(x-m)^2}{2\sigma^2}\right),$$

where σ^2 represents the variance of the noise term. When ensuring self-consistency, we find that the standard form Langevin equation for the equation above has the form:

(6) $$X(t + dt) = X(t) + A(X(t), t)dt + D^{1/2}(X(t), t)N(t)\left(\mathrm{d}t^{1/2}\right)$$

$N(t)$ is a unit normal uncorrelated random variable, and $D(x, t)$ is a smooth, non-negative function. Although the form above is the mathematically rigorous form, we can also write the equation less rigorously, which will prove useful for the market model constructed later in this thesis. The derivative of the Langevin equation technically

cannot be calculated due to the noise term $N(t)$. However, we can image dX/dt exists and alter the equation accordingly [2].

(7) $$\frac{X(t+dt)-X(t)}{dt} = A(X(t),t)dt + \frac{D^{1/2}(X(t),t)N(t)}{(dt^{1/2})}$$

If we define the Gaussian white noise process by:

(8) $$\Gamma(\mathrm{t}) \equiv \lim_{dt\to 0} N(0,1/dt)$$

Then:

(9) $$\frac{dX(t)}{dt} = A(X(t),t) + D^{1/2}(X(t),t)\Gamma(\mathrm{t})$$

Note that Gaussian white noise has an average value of zero and is uncorrelated with itself. Additionally, the integral of a Markov process has a proper integral of the form:

(10) $$\frac{dY(t)}{dt} = X(t)$$

which can be represented as an update formula as well.

(11) $$Y(t+dt) = Y(t) + X(t)dt$$

Note that later in this paper I will work with a Langevin equation for stock market returns, which can then be related to the price movement of the stock via Equation 11. When simulating the particle's movement over many different simulations, it will be useful to calculated an envelope that represents one or two standard deviation. This envelope can be determined theoretically for some processes, however, an analytical result will not be possible for the equation used in this paper.

II. Langevin Market Model Theory

Changes in the market price of a stock are due to differences between the demand for the stock and the supply of the stock.

(12) $$\frac{dx(t)}{dt} = u(t) = \mathcal{F}(\Delta\phi)$$

The price of the stock is represented by the variable $x(t)$. The price will change as a result of changes in supply and demand, based on the effect of function $\mathcal{F}$.

This supply/demand offset, $\Delta\phi$, is equal to demand minus supply.

$$\Delta\phi = \phi_+ - \phi_- \quad (13)$$

The impact of how much a supply/demand offset affects the change in market price is determined by the depth of the market and its ability to simply absorb small offsets temporarily without large changes in the price of the underlying stock, λ.

$$u(t) = \frac{\Delta\phi}{\lambda} \quad (14)$$

This supply/demand offset, $\Delta\phi$, will be broken down into two parts for the purposes of the model used in this paper: (1) absorption of orders by the market, and (2) the spontaneous appearance of new buyers and sellers [1].

In general, the rate at which the market can absorb buy and sell orders is given by:

$$\left|\frac{d\phi_\pm}{dt}\right|_{MM} = -\Theta_\pm(\phi_\mp)d\phi_\pm \quad (15)$$

The equation above illustrates that the absorption of a buy order, for example, is inversely proportional to a function of the number of sells orders, times the magnitude of the buy orders. In this model, we will assume the function Θ is symmetric for supply and demand:

$$\Theta_+ = -\Theta_- \quad (16)$$

To the lowest order, we can model Θ as:

$$\Theta(\phi) = \gamma + \gamma'\phi + \cdots \quad (17)$$

The variable γ represents the liquidity of the market, which determines how quickly a buy or sell order can be absorbed.

We will approximate $\theta(\Delta\phi) = \gamma$, from which we can calculate:

$$(18) \qquad \left|\frac{d\phi_+}{dt}\right|_{MM} - \left|\frac{d\phi_-}{dt}\right|_{MM} = -\gamma\phi_+ - (-\gamma\phi_-)$$

$$(19) \qquad \left|\frac{d\Delta\phi}{dt}\right|_{MM} = -\gamma\Delta\phi$$

The equation above suggests that the rate at which the supply/demand offset will change will be opposite the size of the supply demand offset, and weighted by a liquidity factor, γ, which has the units 1/[time]. In other words, the larger the liquidity of the market, the more quickly supply/demand offsets can be absorbed. As a result, we will see a small deviation from the stock's mean value. It is important to note that the liquidity of the market is variable. Often, liquidity is there only until you actually need it—during market downturns, for instance—at which point liquidity is greatly reduced. As a result, deviations from fundamental stock values are more common and volatility will increase. Therefore, it is worth noting that liquidity may also be influenced by the past market returns [1].

Next we will consider the appearance of new buyers and sellers in the market. This term will incorporate the introduction of new market information in the form a random noise term.

$$(20) \qquad \left|\frac{d\phi_\pm}{dt}\right|_{SP} = m_\pm(R, \psi, t) + \Gamma(t)$$

The average increase/decrease in supply/demand, $m_\pm$, is a function of the expected return, expected standard deviation, and time. The Gaussian white noise term, $\Gamma(t)$, has a mean value of zero, and a variance that is proportional to the market depth, λ, times the susceptibility of the market to random shocks.

$$(21) \qquad \text{var}\{\Gamma(t)\} = \lambda D$$

The anticipated return $R(t)$ is modeled as a constant return plus the trend of recent observed prices.

(22) $R(t) = R_0 + \alpha\bar{u}(t) - k(x - x_0)$

(23) $\alpha = a - a'\bar{u}(t)$

Alpha is the weighting of the recent price trend on anticipated returns. Investors are risk adverse: Negative returns weigh more heavily in their minds than positive returns. Therefore, α will be larger when the recent returns have been negative. The final term in the expected return equation describes mean reversion to an average stock price value, which of course will change as a function of time. The coefficient, k, describes how quickly this mean reversion will occur. For our purposes, we will assume k is negligible since we will be modeling the price over short time scales.

The recent price trend is calculated using a normalized Kernel, K_R, which weights historical return values. In this paper, we will use a simple weighting of historical model returns.

(24) $\bar{u}(t) = \int_0^t dt' K_R(t - t')u(t')$

Similar equations are used for the expected volatility in the model.

(25) $\psi(t) = \psi_0 + \beta\overline{u^2}$

Once again, β is the influence of past volatility on the market's expectation of future volatility. A normalized Kernel is also used to weight historical volatility values

(26) $\bar{u}(t)^2 = \int_0^t dt' K_\psi(t - t')[u(t')]^2$

Assuming $m_\pm$ is a linear function, we can expand it to lowest order:

(27) $m_+ = m_{0+} + \alpha_+\bar{u} - \beta_+\overline{u^2}$

(28) $m_- = m_{0-} + \alpha_-\bar{u} - \beta_-\overline{u^2}$

We can then calculate the rate of change of spontaneous supply demand offsets:

$$\left|\frac{d\phi_+}{dt}\right|_{SP} - \left|\frac{d\phi_-}{dt}\right|_{SP} = (m_{0+} - m_{0-}) + (\alpha_+ - \alpha_-)\overline{u} - (\beta_+ + \beta_-)\overline{u^2} + \Gamma(t) \quad (29)$$

$$\left|\frac{d\Delta\phi}{dt}\right|_{SP} = m_0 + (a - a'\overline{u})\overline{u} - \beta\overline{u^2} + \Gamma(t) \quad (30)$$

Finally, we can calculate the total rate of change of the supply/demand offset:

$$\frac{d\Delta\phi}{dt} = \left|\frac{d\Delta\phi}{dt}\right|_{MM} + \left|\frac{d\Delta\phi}{dt}\right|_{SP} = -\gamma\Delta\phi + m_0 + (a - a'\overline{u})\overline{u} - \beta\overline{u^2} + \Gamma(t) \quad (31)$$

The equation above relates the supply/demand offset to market liquidity, influence of past returns on future returns, influence of past volatility, and random noise resulting from new information about the economy. We now want to relate this equation to the rate of change of the stock's price, $u(t)$. From Equation 12 above, we know that the rate of return is equal to the supply demand offset divided by the market depth. Therefore, we will multiply the above equation by $1/\lambda$. We will also assume the average offset, m_0, is equal to zero—in other words, the model does not have drift [1].

Having taken into account the simplifications described above, we finally come to the Langevin equation used in this paper:

$$\frac{du(t)}{dt} = -\gamma u(t) + (1/\lambda)(a - a'\overline{u})\overline{u} - (1/\lambda)\beta\overline{u^2} + (1/\lambda)\Gamma(t) \quad (32)$$

We will model this equation using the update formula described above (Equation 6). Each return value will be calculated using the previous return value, plus an incremental slice of the derivative of return over a small time step.

$$u(t + dt) = u(t) + \left[-\gamma u + \left(\frac{1}{\lambda}\right)(a - a'\overline{u})\overline{u} - \left(\frac{1}{\lambda}\right)b\overline{u^2} + \left(\frac{1}{\lambda}\right)\Gamma(t)\right]dt \quad (33)$$

III. Model Implementation

The Stochastic Differential Equation (SDE) in Equation 33 was simulated in Excel VBA, C++, and MATLAB, and as I transitioned from Excel to MATLAB, an

increased level of sophistication was used. Initially, the equation was modeled simplistically using only a drift term and a unit normal Gaussian white noise as the fluctuating force. Multiple trials were performed for this data and stored in an array. The standard deviation of the particle about the mean was calculated in order to quantify the diffusion envelope of the particle. Analysis was performed to determine the appropriate relationship between the drift and diffusion terms, such that the desired magnitude of fluctuations could be achieved. At this stage, the model did not possess value in understanding the movement of the stock market, however, it offered the opportunity to understand the relationship between variables in simple Markov random walks [8].

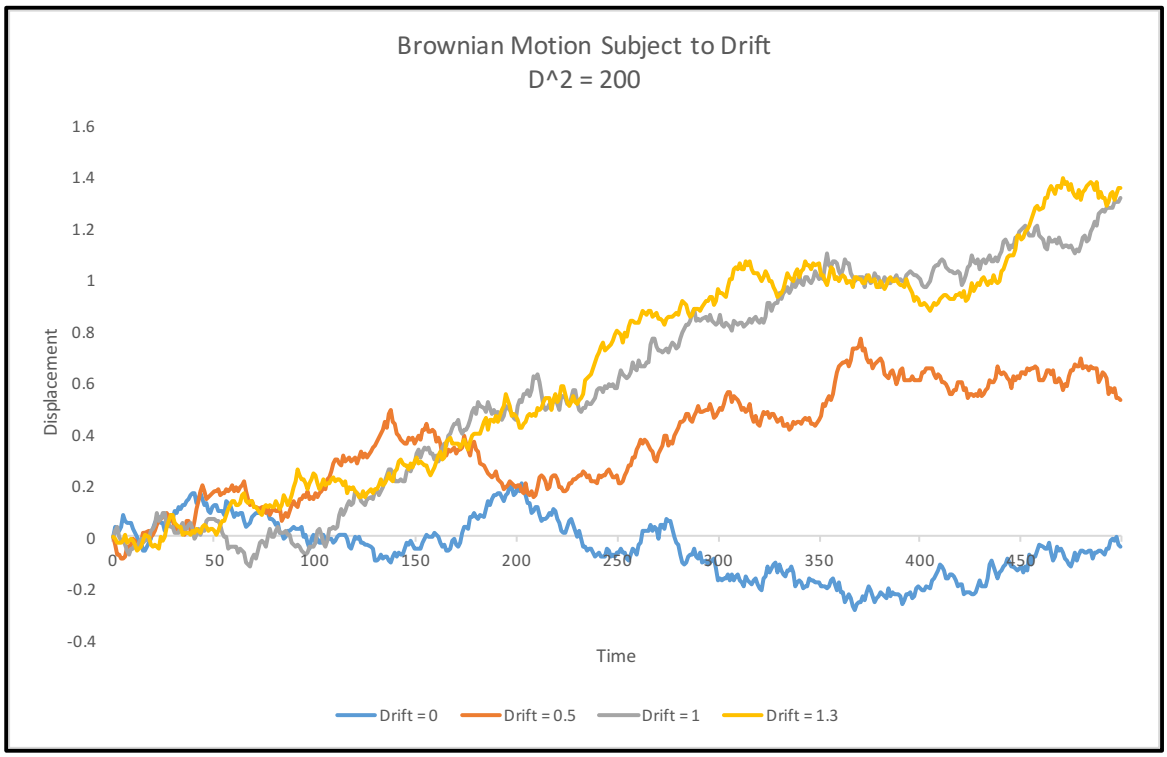

Figure 1: Random Walk with Drift ($\sigma = 1$; $D^2=200$; 500 Steps)

In this simulation, the Equation 9—the update formula—was used. A unit normal Gaussian random term was incorporated, and 500 steps were simulated. In order to more accurately reflect the behavior of the curves, 1000 simulations were performed with each

drift coefficient. The average of those simulations for each set of coefficients has been plot in the figure above. The fluctuation term remains constant throughout the simulations, however, as a result of the persistent positive drift term, the particle tends to move away from the x-axis. The blue curve represents an unbiased, uncorrelated random walk. It is clear that although the particle will fluctuate, its expectation value over time will remain zero.

In the model used in this paper, no bias has been introduced to the equation. Since we are modeling the stock price *returns*, we would not want the returns persist too long in the positive or negative direction. However, we do introduce memory terms in order to allow positive or negative trends to continue for a period of time, until the effect of liquidity pulls the market to a mean return of zero. Over the long term, the expected return is greater than zero, however, for simplicity we will not include a positive bias term.

The standard deviation of the market return is given by:

$$\langle u^2(t+dt)\rangle = \langle\left(u(t) + \left[-\gamma u + \left(\frac{1}{\lambda}\right)(a - a'\overline{u})\overline{u} - \left(\frac{1}{\lambda}\right)b\overline{u}^2 + \left(\frac{1}{\lambda}\right)\Gamma(t)\right]dt\right)^2\rangle \tag{34}$$

We will instead calculate the standard deviation of the rate of return numerically, as there is no analytical solution for the above equation.

The Langevin equation used to model the market has five coefficients that need to be fit to market data in order to provide an equation that will fit specific market curves. Initially, a full five-coefficient model was built in order to estimate the coefficients, although different models were used in order to allow for faster computation times. In order to determine which sets of coefficients were most accurate, one method used was a paired comparison test. The method subtracts one value of a pair of data points from the

other at the same point in time, and then calculates test statistics based on these differences.

(35) $$\bar{d} = \frac{\sum_{i=1}^{n} d_{market,i} - d_{model,i}}{n}$$

In the equation above, I calculated the average difference of all the points being compared.

(36) $$S_d = \sqrt{\sum_{i=1}^{n} \frac{(d_i - \bar{d})^2}{n-1}}$$

(37) $$t = \frac{\bar{d}}{S_d/\sqrt{n}}$$

The standard deviation of the differences was then calculated and used in Equation 37 to calculate the t-score for the paired comparison test. We would expect good fits to result in t-scores close to zero.

The Langevin equation was fit to ten different sets of intra-day S&P 500 Index market data. Using the NYSE website, I collected index price information every five minutes between March 21 and April 5 [6]. I chose five-minute intervals for two main reasons. Most importantly, over this short time period it is unlikely that fundamental changes to the market will take place, thus allowing me to approximate the mean spontaneous appearance of buys and sellers as zero. Additionally, this period range created 78 return data points for each day of trading analyzed, which was small enough that the Langevin simulations could be complete in a reasonable amount of time. Had one-minute data been chosen instead, we would have had five times as many data points to simulate, which would have increased the computation times by an order of magnitude.

The intra-day data was stored in an Excel file and imported to MATLAB. Using the built in *tick2return* function, these price movements were converted to return information and stored as MATLAB variables, which were saved and could be used at a

future date to run simulations. Additionally, I calculated the variance and geometric mean of each data set.

After loading the desired market data into the Langevin script, I determined the number of steps to be performed in the model based on the number of steps in the market data. Because I am modeling a stochastic differential equation, an individual trial is heavily dependent on the random term, which has been modeled as Gaussian white noise. In order to gather more meaningful results, I averaged over numerous simulations for a given set of assumptions. In the majority of the coefficient sets analyzed, 1000 simulations were used. This number was largely determined based on computational limitations. Because each set of coefficients needed to be simulated 1000 times, very quickly the program execution time became on the order of hours or days. When testing fewer coefficient sets it was possible to perform more simulations for each set, which increased the statistical soundness of the analysis.

The coefficients to be tested were stored in arrays constructed using linear spacing between a value of zero and a final coefficient value determined by the user. The number of coefficients stored in the array was the largest determinant of program execution time. Increases in the number of points tested for each coefficient resulted in the execution time increasing as a power law.

The noise term used in the model was determined at the outset using a large seven dimensional matrix. Using the *normrnd* MATLAB function, values were calculated for each set needed as I iterated through the different coefficient combinations. The random value from the matrix was tracked using the FOR loop iterators, which were labeled accordingly.

A number of FOR loops were used to ensure a set of data was constructed for each unique set of coefficients. Five FOR loops represented the different coefficients being tested, and an additional two allowed me to first simulate the equation, and then repeat the simulation a set number of times, resulting in a total of seven FOR loops during the model market data phase of the analysis.

The update formula used to model the market data (described in Equation 33 above), incorporates five coefficients that model the market behavior:

$\boldsymbol{\alpha}$: Recent return trend's influence on future returns

$\boldsymbol{\alpha'}$: Recent return trend's influence on future returns when the past return is negative (represented risk aversion)

$\boldsymbol{\beta}$: Recent volatility of returns trend's influence on future returns

$\boldsymbol{\lambda}$: Market depth (impact of new information on return fluctuations)

$\boldsymbol{\gamma}$: Market liquidity (determines how quickly market returns to equilibrium after deviation from mean)

One of the most significant terms is λ, given that it controls how much each incremental time step is weighted when the simulation is evolved. Additionally, γ plays a significant role bringing the return value to a mean value of zero, offsetting the previous return value. This prevents the model from deviating significantly from a return of zero.

III (a): Approximating Lambda

The Langevin equation models the impact of new information on the supply and demand as a random process with a mean effect of zero. Over time, the new information cancels itself out. Of course, this may not be an accurate assumption, however, due to the short time periods were are dealing with in this model, we do not need to worry about

longer term changes in price. The random impact of new information is modeled as Gaussian white noise. Lambda represents the market depth, or the impact of supply and demand offsets on the price. If lambda is large, the result of new information on the rate of change of the stock price will be small.

Lambda was approximated using a simplified Langevin equation that only incorporated the liquidity factor and the random noise term. First an array was created using the *linspace* function in MATLAB between a starting value of 1e-4 and a final value of 12e-4. This range was determined after numerous preliminary sets on the sample data. The corresponding gamma values tested ranged between 0.9 and 1.1. The purpose of including gamma was to take into account that fact that some of the volatility is due to the liquidity factor, not the random noise of new information. For each unique combination of lambda and gamma 1000 simulation were performed. An average standard deviation was calculated over all the simulations for the given initial conditions. This average was then compared to the calculated standard deviation of the empirical market data being analyzed. An F Test was used to determine which set of values for lambda produced the most accurate fit to the empirical data.

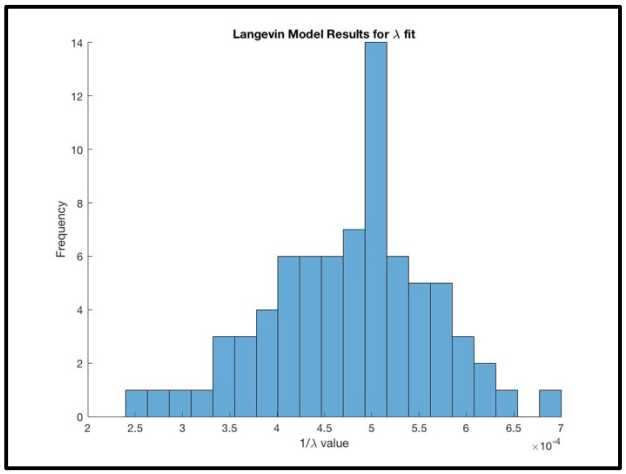

Figure 2: Histogram of Lambda Values using F Test Comparison
Average computation time: 93.386 s

III (b): Approximating Gamma

Gamma represents the liquidity of the market and the ability of the price of the index to revert back to its previous value after a supply/demand offset. In the Langevin equation, $-\gamma u(t)$ caused the curve generated by the model to trend back to a value of zero, having deviated slightly during the previous time step due to the effect of the noise term. Depending on the value of gamma, the curve will either revert back to zero quickly or slowly. In the empirical data, one can see the "jitteriness" of the return values; gamma accounts for much of this effect in the Langevin equation.

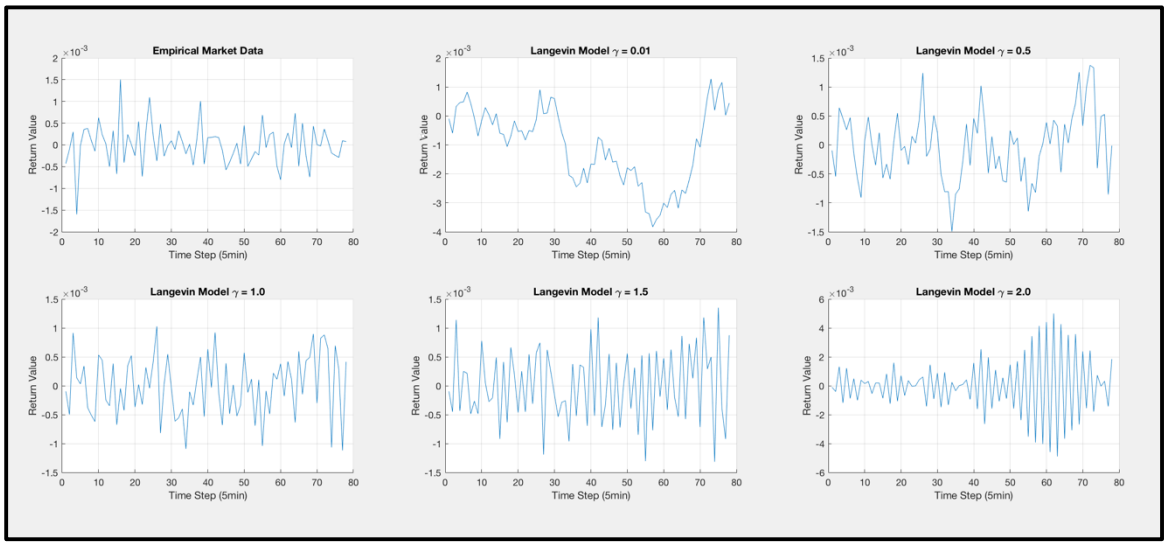

Figure 3: Effects of Gamma in Langevin Model

In the figure above, I have varied the value of gamma in the Langevin model to demonstrate the effect on the curves generated. When gamma is very low, the curve can deviate significantly from the mean value of zero, given that there is very little "pull" back toward zero. Although over a large number of steps the curve will remain roughly around zero since there is no drift, over smaller time periods the deviation from zero can be significant. As gamma is increased, the oscillation about the x-axis increases, and the duration of the deviation from zero decreases. Based on visual inspection of the curves,

it is apparent that a gamma value of between 0.5 and 1.0 seems appropriate for the model approximating the market data set included in the upper left hand corner of the figure.

An unexpected effect manifested itself at very large gamma values. In the bottom right figure gamma is very large, approximately 2.0. This means that any deviation from zero will result in a next step that is twice the previous deviation, but in the opposite direction. As a result, the volatility increases, which one would expect. Additionally, however, a certain periodicity seems to appear as a beating phenomenon with a characteristic period of oscillation. Although this phenomenon was not explored in this research, future analysis may explore whether during market periods with large gamma values it may be possible to accurately time the market swings due to the periodicity of the beating effect.

Having successfully determined a value for lambda, I then use the same equation and instead focus on a wide range of gamma values. One hundred possible gamma values were tested for each market data set, with one thousand simulation performed for each unique gamma value. In order to determine which value most accurately matched the market data, I calculated how long the curve remained in the positive or negative return region, and how often the generated curve crossed the x-axis. This information was compared to the calculated values for the market data set. One way I could improve our analysis would be to calculate the total area underneath the curves, to take into account the magnitude of the deviations.

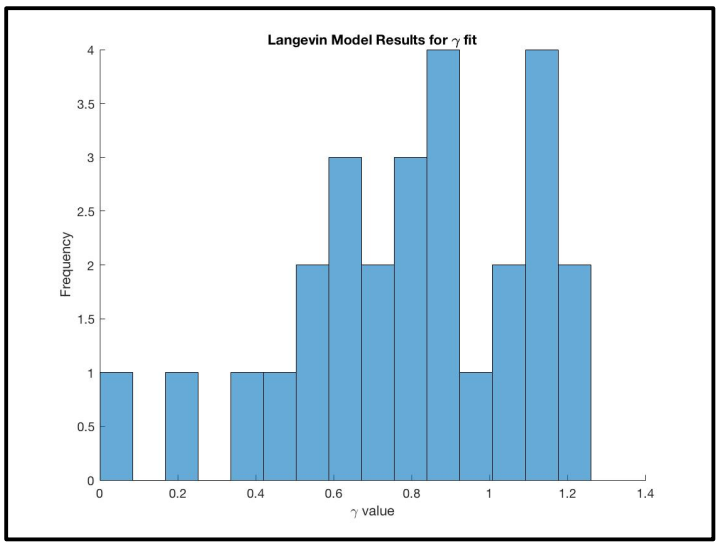

Figure 4: Histogram of Gamma Values using Average Deviation from Zero comparison

The figure above visually illustrates the results from a simulation used to determine gamma for a data set. After the length of deviation from zero was calculated for the model data set, it was compared with the market data set. If the results were in good agreement, the gamma value was stored in a matrix. After all combinations of gamma values were tested, the matrix was plotted as a histogram, which allowed the user to see which gamma values were accurate approximations of the market data. Additionally, the underlying matrix that was plotted in the histogram also contained data that allowed the user to compare the accuracy of the gamma values that fell within the desired range. In the case of a tie, like the simulation above, the more accurate fit was chosen from this matrix.

Average computation time: 55.374 s

III (c): Approximating Alpha and Beta

The terms alpha and beta take into account the effects of recent trends on future returns. Alpha incorporates the recent return values over a window of ten data points on future returns. The weightings were calculated using MATLAB's moving average

function and a simple weighting. Alpha determines how much recent returns will impact future returns. Additionally, the effect of the volatility of recent returns was also calculated and weighted as a moving average with a window of the past ten return values. In the Langevin equation, $-b\overline{u^2}$ is always negative, which means that larger volatility in the returns always causes the volatility term to pull down future expected returns in the negative direction. Investors do not like uncertainty. Beta determines how much of an impact recent return volatility should have on future returns. For each new step calculated, the moving average function must be called in order to determine what the recent return trend has been. As a result, the program execution time increases tremendously as compared to the simulations performed to determine lambda and gamma. [Note: A separate moving average of the variance of the past ten returns must also be calculated, further contributing to an increase in program execution time.] Lambda and gamma are both taken to be constants, given that they have been determined previously. In order to limit the time to execute the program, only one hundred simulations were performed for each unique combination of alpha and beta.

Twenty alpha values and twenty beta values were tested for each data set, for a total of 400 unique combinations, and 40,000 curves generated. The optimal alpha-beta combination was determined by comparing the geometric mean of the model data sets to the geometric mean of the market data. I chose the geometric mean—as opposed to the arithmetic mean—because it takes into account the effect of losses on the return potential of a portfolio due to the destruction of principal. In effect, the alpha and beta terms allow the Langevin model to trend in one direction or another and not immediately revert back

to a value of zero, thus allowing for longer term price movements of the underlying stock being modeled, despite the mean of the noise term being zero.

(38) $$geomean = (\prod_{i=1}^{n} x_i)^{\frac{1}{n}}$$

A particular difficulty of modeling alpha and beta was the sensitivity of the model to small deviations once a critical value was reached. The model would quickly increase or decrease exponentially if the alpha or beta values were too large. As a result, these data sets would become corrupted and include "not-a-number" values, which could not be processed by later function used, which would cause the code to fail. Therefore, it was important to screen the results at this stage and remove data sets that were obviously too large and a result of the model's excessive sensitivity.

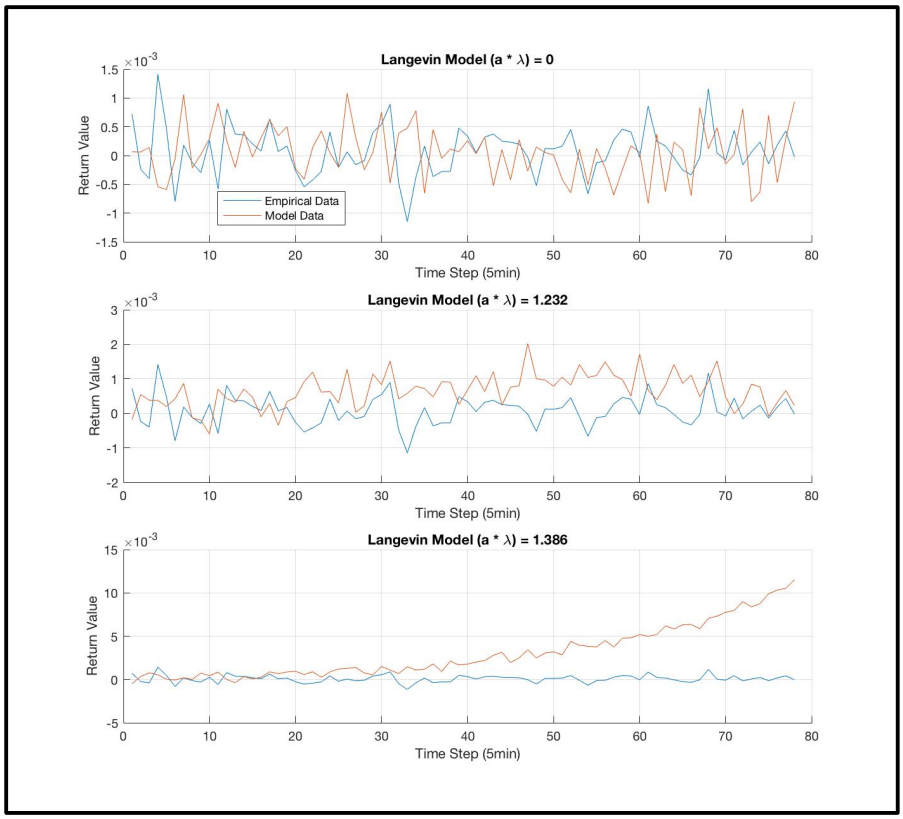

Figure 5: Effects of Alpha in Langevin Model

The figure above illustrates the effect of changes in alpha relative to gamma on the behavior of the Langevin model. As you can see from the middle graph, larger alpha

values allow the curve to exist above or below the x axis for a significant period of time, which produces larger geometric mean results. However, when alpha is larger relative to lambda, the effects of memory are too pronounced, and a slight deviation may quickly turn into an overwhelming tendency towards positive or negative returns. It is just as likely the model would produce large negative returns as large positive return given a significant alpha value relative to lambda.

Similarly, changing the coefficient beta is also an effective way to alter the geomean of the model, however, beta is even more sensitive to its relative weight to lambda than is alpha.

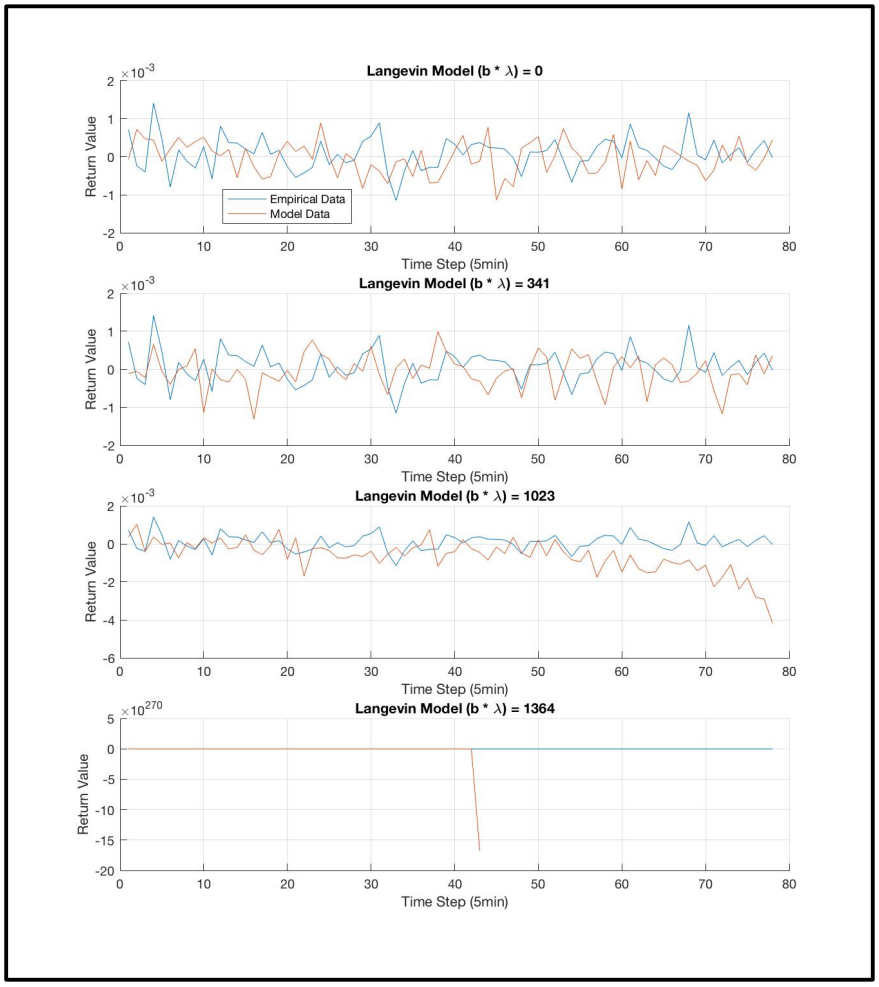

Figure 6: Effects of Beta in Langevin Model

Since the beta term is always negative, larger beta values pull the model in the negative direction. Additionally, there is a very small range over which the beta values are significant enough to effect the model, yet not too large to become overwhelming influential. As one can see in the bottom graph in Figure 6, a large beta value relative to lambda produces nonsensical results. Therefore, it was important to have beta occupy a someone tightly constrained range. This range was assumed to be constant throughout all the data sets analyzed, however, it may be beneficial to vary the range given significant changes in the underlying market data.

Average computation time: 458.2 s

IV. Results & Discussion

The market data was first analyzed to determine how long the returns deviated from zero, the volatility, and the geometric mean. In Table 2 below, it is apparent that the volatility of S&P returns over the period observed remained on the order of 10^{-7}. As a result, we expected the $1/\lambda$ values to all be of the same order of magnitude with each other and to not vary too significantly between each data set tested. The next interesting observation was that the average length of time over which a given positive or negative trend persisted was tightly clustered around one time step (5 minutes). As a result, we do not expect significant returns in the positive or negative direction. This was further observed when the geometric means of the data sets were calculated. There was a somewhat wide range of geomean values for the different samples tested. Typically, the geometric mean was on the order of 10^{-5}, however, some data sets has more significant mean values, indicating that positive or negative trends tended to persist longer than the other data sets.

Using the MATLAB code from Appendix A, multiple simulations were used to determine the best fit for alpha, beta, gamma, and lambda for each data set using Equation 33. Note that it was decided to eliminate the risk aversion term, a', that gave more weight to negative return trends than positive price trends. This allowed the simulation times to be faster, and I believe that over the short time periods examined, the risk aversion term would not be very significant. However, with increased computer power, it would be beneficial to reintroduce the risk aversion term.

Lambda and gamma terms were both determined prior to alpha and beta, give that they had the most significant impact on the overall behavior of the model curve, whereas alpha and beta refined the curve and made the model more accurately fit the given data set one the broader curve features were already taken into account. In order to determine whether the lambda and gamma values were good overall fits compared to the market data, the Langevin model was simulation 1000 times with the specific lambda and gamma values. At each time step, the standard deviation of all 1000 simulation was calculated and stored in a matrix. The standard deviation band produced through the model was then plotted on the same graph as the market data in order to see if the estimates were roughly accurate.

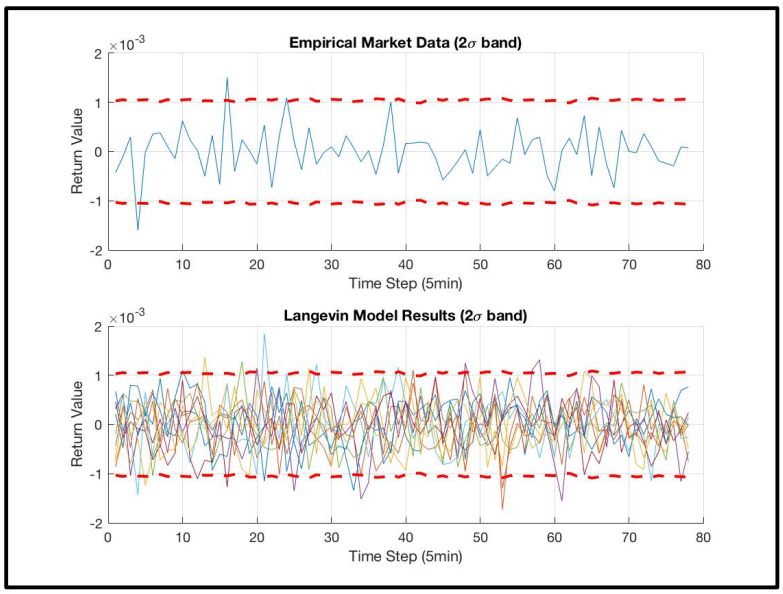

Figure 7: Two Standard Deviation Envelope Comparing Langevin Model to Market Data

As you can see in Figure 7 above, the majority of the market data is captured between the two standard deviation bands, which show the range of results from the Langevin model, suggesting the current values for lambda and gamma are roughly accurate.

The computationally difficult portion of the model was approximating the alpha and beta values. The difficulty lies in the historical moving averages that must be calculated for each step taken for both the return and the square of the return. As a result, MATLAB must call the *tsmovavg* function millions of times, which significantly delays the program. This part of the model is integral to the ultimate results, however. By varying the impact of recent trends on future return values we can alter the geometric return of the model and more accurately simulate the market data set.

The geometric mean of each simulation was calculated and compared to the market data geometric mean in order to determine goodness of fit. We found that the alpha and beta values typically have a somewhat large range of values that may have produced relatively accurate results.

Table 1: Coefficient Results fitting Langevin Model to Market Data

	α	β	γ	$1/\lambda$
21-Mar-16	815.79	1.06E+05	1.1742	4.78E-04
22-Mar-16	1210.50	1.48E+05	0.9343	5.22E-04
23-Mar-16	973.68	1.27E+05	0.8081	5.00E-04
24-Mar-16	215.49	1.39E+04	0.8460	4.80E-04
28-Mar-16	1613.20	1.13E+04	1.0606	4.56E-04
29-Mar-16	1757.89	1.39E+04	0.9596	5.78E-04
30-Mar-16	500.00	5.34E+04	0.9848	4.89E-04
31-Mar-16	736.84	1.69E+05	1.0480	4.22E-04
1-Apr-16	1760.00	2.16E+04	0.9722	5.67E-04
4-Apr-16	1290.00	8.74E+03	1.0606	4.67E-04
5-Apr-16	736.84	9.53E+04	0.9091	6.33E-04

Table 1 tabulates the results of fitting the four coefficients in the Langevin model to the different market data sets.

Table 2: Market Data values for a variety of quantitative measures

	Average Length	**Volatility**	**Geometric Mean**
21-Mar-16	0.71	2.27E-07	2.17E-05
22-Mar-16	0.90	2.68E-07	3.88E-05
23-Mar-16	1.05	2.43E-07	-5.39E-05
24-Mar-16	1.29	1.85E-07	8.13E-05
28-Mar-16	0.81	2.01E-07	-1.36E-05
29-Mar-16	1.00	3.26E-07	1.47E-04
30-Mar-16	1.17	2.35E-07	-1.03E-05
31-Mar-16	0.65	1.78E-07	-3.06E-05
1-Apr-16	1.11	3.23E-07	1.64E-04
4-Apr-16	0.85	2.16E-07	-2.84E-05
5-Apr-16	0.73	3.91E-07	-3.45E-05

Table 2 analyzes the market data sets based on the three metrics that were used significantly in the Langevin model to test goodness of fit.

Comparing the results from the Langevin coefficient fit (Table 1) to the quantitative measures in of the market data (Figure 2) we can see that the lambda values

are very accurate. The data set with the largest volatility—April 5—also has the largest $1/\lambda$ value at 6.33E-4, which would be expected. Alpha and beta allow return trends to persist, and therefore the market data sets with the largest geometric means would be expected to result in relatively large alpha values in order to generate greater than zero average returns. We find that March 29 and April 1, which have the two largest magnitude geometric mean values, both resulted in the largest alpha values. This suggests that return trends are an effective way to produce average return results that are in line with the geometric mean of the underlying data set, despite the mean of the Gaussian white noise term being zero.

Table 3: Geomean of Langevin Model versus Market Data for March 29 (Sample of Process)

a Values Tested \ b Values Tested	1.00E+03	1.10E+04	2.09E+04	3.09E+04	4.08E+04	5.08E+04	6.07E+04	7.07E+04	8.06E+04	9.06E+04	1.01E+05	1.10E+05	1.20E+05	1.30E+05	1.40E+05	1.50E+05	1.60E+05	1.70E+05	1.80E+05	2.00E+05
50.00	0.064	-0.056	-0.096	-0.028	0.010	-0.019	0.047	-0.037	0.044	-0.152	0.014	-0.039	-0.059	-0.117	-0.119	-0.009	-0.039	-0.107	0.019	-0.058
147.50	0.028	-0.020	-0.005	0.035	0.045	-0.177	-0.028	-0.027	-0.021	-0.028	0.049	-0.103	-0.038	0.056	0.013	-0.047	-0.007	-0.068	-0.121	0.005
245.00	0.024	0.043	-0.089	-0.056	0.167	-0.010	-0.034	-0.101	0.047	0.041	-0.077	-0.024	-0.161	-0.005	-0.025	-0.020	0.023	-0.145	-0.074	-0.003
342.50	0.099	0.034	-0.080	-0.031	-0.047	0.125	-0.020	0.024	-0.019	-0.133	-0.026	-0.081	-0.094	-0.163	-0.156	-0.039	-0.045	-0.027	-0.090	-0.119
440.00	-0.016	-0.104	0.075	0.071	0.013	-0.118	0.057	-0.023	-0.052	-0.190	-0.068	-0.018	0.002	0.002	-0.104	-0.120	-0.098	-0.115	-0.006	-0.086
537.50	-0.080	-0.005	-0.148	-0.031	0.117	0.153	0.006	-0.243	-0.036	-0.071	-0.104	-0.191	0.041	-0.093	-0.063	-0.093	-0.147	-0.219	-0.077	-0.163
635.00	0.006	-0.210	0.148	-0.030	-0.033	-0.090	-0.075	-0.210	-0.116	-0.065	-0.008	0.040	-0.133	-0.207	-0.035	-0.348	-0.098	-0.151	-0.296	-0.207
732.50	0.061	0.021	0.051	-0.047	0.030	-0.103	0.084	0.077	-0.351	-0.020	-0.134	-0.130	-0.207	-0.292	-0.096	-0.078	-0.175	-0.031	-0.270	-0.021
830.00	-0.138	-0.049	0.329	0.152	-0.039	-0.087	0.018	-0.339	-0.129	0.033	-0.116	-0.383	0.023	-0.284	-0.131	-0.361	-0.082	-0.255	-0.164	-0.233
927.50	-0.011	-0.009	-0.198	0.098	-0.150	-0.005	-0.180	0.335	0.238	0.026	-0.098	-0.329	-0.095	-0.066	-0.301	-0.670	-0.194	-0.059	-0.333	-0.262
1025.00	-0.066	0.096	-0.600	0.140	-0.555	-0.199	-0.061	-0.465	0.264	-0.092	-0.687	-0.365	-0.328	-0.274	-0.163	-0.474	-0.251	-0.494	-0.480	-0.287
1122.50	0.474	-0.014	-0.232	0.322	-0.632	-0.661	0.047	0.026	0.235	-0.160	-0.774	-0.378	-0.448	-0.180	-0.988	-0.678	-1.084	-0.509	-1.065	-0.417
1220.00	-0.006	-0.014	-0.562	-0.318	0.081	0.449	-0.330	-1.714	0.793	0.442	-1.362	-0.411	-2.166	-1.366	-1.049	-0.767	-1.418	-0.957	-1.392	-2.066
1317.50	-1.439	3.377	1.332	0.720	-1.844	-1.096	-0.932	-0.907	-1.215	-2.583	-0.357	-1.462	-4.070	-3.073	-3.452	-4.698	-3.496	13574.00	6785.116	-3.917
1415.00	-2.879	-1.689	-3.665	-4.685	-0.147	-2.383	-2.197	-3.427	6782.599	6785.980	13574.50	20353.91	13562.50	6778.34	54312.35	61093.00	61092.75	67879.56	54307.97	101836.2
1512.50	-8.959	-5.189	-7.034	-9.793	-8.839	20359.52	27137.8	67876.8	54294.2	88256.1	88251.4	108618.4	122190.9	115411.7	169726.0	210460.9	196885.8	203682.6	169725.7	183305.6
1610.00	0.423	6768.302	40728.8	67878.9	74674.2	88242.1	190098.8	176509.5	183308.4	230843.7	210470.8	217262.5	230844.2	224047.5	251211.6	251205.4	298726.2	291947.5	285158.6	291946.1
1707.50	30.164	101847.3	128998.9	169751.9	169767.9	251216.0	176556.9	210475.5	278377.8	298751.1	258016.2	278382.5	353059.4	278380.2	359843.9	312322.4	285171.7	312320.9	325899.5	319112.2
1805.00	95085.6	183413.7	244467.0	305589.5	353090.2	251264.1	285202.3	353088.1	332716.9	373441.7	325922.5	393805.8	312343.8	244457.5	325921.0	339490.0	319126.2	298760.4	353068.8	291969.0
2000.00	210729.7	271749.2	326016.3	298859.1	353130.9	285241.5	373473.0	292019.8	319169.0	366676.6	278423.9	325933.6	346294.2	407390.0	359874.3	380234.9	285196.5	387023.1	319134.7	366653.7

In the above table I calculated the ratio of the geometric mean of the Langevin model divided by the market data geomean. The closer this ratio was to one or negative one, the more accurate the combination of alpha and beta was assumed to be. A FOR loop captured the corresponding alpha and beta values that met the desired range of accuracy.

The Langevin model constructed allows the user to determine the magnitude of return oscillations, the speed with which the return reverts back to zero, as well as the impact of past return trends on future return values. By breaking apart the impact of these components on the Langevin model predictions, we can better understand the source of the movement of stock market prices. From a simulation perspective, a number of improvements could be made in order to produce even more accurate results. Having constructed the theoretical basis of the Langevin model, the most important function was determining the goodness of fit for a given set of coefficients. I tried a variety of different methods, settling on the ones used in this paper. However, I believe even more effort is needed in order to determine an optimal set of coefficients that are robust. First, I believe the number of simulations used for the alpha and beta fits was inadequate. Only

one hundred simulations were performed per set of alpha and beta values. In contrast, 1000 simulations were performed during the lambda and gamma calculations. This was due to the computation time increase for the alpha/beta simulation, which was a result of the incorporation of memory into the Langevin equation. Given increased computer power, I believe it would be more accurate to run between 1,000 and 10,000 simulations per combination. This would have caused simulation times to take over 24 hours, which was not an option given available resources.

As seen in the Figure 7, a two standard deviation band was placed around market data, allowing us to see when deviation exceeded the anticipated range. From a risk management perspective, if the portfolio deviated beyond this range, it may suggest that the model is no longer an accurate reflection of the market conditions. Alternatively, if one was confident in the model and believed the underlying conditions had not changed appreciably, one could bet against the market when it deviated beyond the two standard deviation range, in which case one would profit it the historical bounds held.

In particular, I believe the estimate of gamma could be improved by calculating the area under the return curve when it was above or below the time axis. In the model used in this paper, the time period between successive intersections with the time axis was calculated. This fails to account for the magnitude of the deviations. The justification for the simplification was that lambda is supposed to capture the majority of the volatility in the model, however, of course the volatility is dependent on multiple factors. Incorporating gamma into the model's approximation of market volatility would improve accuracy.

The theoretical Langevin model used to simulate market data (Equation 33) may be improved by keeping the results more general and not making a number of marked simplifications. Most significantly, the theoretical model assumed that market makers and spontaneous buyers/sellers acted the same way on the buy and sell side of a trade. As a result, I was able to easily calculate the difference between the supply and demand, which allowed me to reach an equation that could be simulated with five coefficients. However, it may be more appropriate to acknowledge that the buy side of a market can become extremely cautious—or euphoric—during certain market conditions, which would cause the execution time for a trade to change appreciable. For example, during tumultuous markets, market makers carry more risk when making short term purchases from sellers given that the stock price may drop significantly before a buyer can be found. As a result, bid-ask spreads increase and the willingness of market makers to facilitate trading decreases as a result of this increase in risk. The Langevin model constructed in this paper is likely only appropriate during stable market conditions.

A simple moving average of past return values was used to calculate the current return trend in the Langevin model. Given the short term nature of the market data, this was believed to be an appropriate technique for weighting the past values and their influence on future model results. It may be interesting, however, to examine the different alpha and beta values calculated given an exponential moving average of past return values.

V. Conclusions

Using a non-linear Lanvevin model, I approximated the behavior of the S&P 500 Index over a trading period of one day, for a variety of different data sets. The four main features of the model were: (1) market depth, (2) liquidity, (3) random introduction of

new buyers/sellers, and (4) system memory. In Table 1, I tabulated the different weightings applied to the terms in the Langevin equation in order to produce accurate curves matching different market data sets. We found that over the two weeks analyzed, the volatility of the market data remained somewhat constant, and the 1/lambda term was typically on the order of 10^{-4}. We also observed that more rapid price oscillations were the result of greater liquidity in the market, as modeled by the gamma term in the Langevin equation. Further refinements to the model were based on adjusting the impact of past market movements on future returns. The coefficients alpha and beta allowed the impact to be weighted appropriately, which led to more accurate geometric mean results. Additionally, these factors created positive and negative average values, when otherwise one would have expected a mean value of zero, given the lack of a drift term in the model.

Physical systems—such as the drift of a colloidal particle in a fluid—can be descried very accurately by a Langevin equation, and rigorous proofs can be used to relate the various variable in the equation to each other. For example, a fluctuation-dissipation relation can be constructed that relates the frictional loss to the fluctuations experienced by the particle.

(39) $$m\frac{dV(t)}{dt} = -\gamma V(t) + F(t)$$

The equation above describes the simple motion of a colloidal particle subject to a fluctuating force, $F(t)$, and a dissipative drag for, $\gamma V(t)$. This equation is based on Newton's Second Law. The drag coefficient can be further described below:

(40) $$\gamma = kT/D$$

The dissipation term, γ, is a function of the solution's temperature and specific diffusion characteristics.

$$F(t) = \left(2kT\gamma\right)^{1/2}\Gamma(t) \tag{41}$$

After a bit of math, one can relate the dissipation term to the fluctuation term. This signifies the analytical relationship between the two terms, and their fundamental connectedness. This result makes intuitive sense given that molecular collisions with atoms are both the cause of the drag as well as the cause of the fluctuation of the colloidal particle. Unfortunately, such a beautiful equation with simple relationships between terms is not available for the Langevin model constructed in this thesis. There are myriad sources that drive the market fluctuations and the appearance of buyers and sellers.

The methods used to determine goodness of fit between the model and the market data were one of the most important aspects of the model implementation. A variety of different methods were used to test the accuracy of coefficients in the model. Specifically, the F test and paired comparison test were two methods that allowed me to distinguish if the model variance and mean deviated significantly from the market data. Comparing the geometric means of the model and market data also allowed me to refine the estimates of alpha and beta. Additional research on the accuracy of the different methods used to assess goodness of fit is recommended. Given increased computing power, it would also be useful to assess a wider range of possible coefficient values, as well as incorporate additional terms—such as risk aversion—that had to be removed in the interest of reducing computation time.

The Langevin model implemented was in good agreement with the market data sets to which the model was fit. The volatility parameters estimated produced

appropriate standard deviation bands which encompassed the market data volatility accurately (Figure 7). Additionally, one could assume that deviations that exceeded the band were anomalies, and would soon revert back to the mean. We found that market data sets that had more significant geometric means—March 29 and April 1, for example—resulted in model coefficient values for alpha being the highest. As a result, we believe the persistence of recent price trends is a significant source of return values over short time periods. However, structural changes in the underlying economics of a country or the world may have effects that are completely unpredictable by this Langevin model. It may be useful to choose to exit the market when the Langevin model estimates become increasingly erratic over time, which would suggest that the market is entering a period of unusual turbulence.

References

[1] J Bouchaud, R Cont, *A Langevin approach to stock market fluctuations and crashes*, Eur. Phys. J B **6**, 543-550 (1998).

[2] D Gillespie, *The mathematics of Brownian motion and Johnson noise*, Naval Air Warfare Center, China Lake. Am. J. Phys. **64** (3), March 1996.

[3] A. Einstein, *Investigations on the Theory of Brownian Movement*, Ann. Phys. 17, 549-560 (1905).

[4] D Gillespie, *Fluctuation and dissipation in Brownian motion*, Am. J. Phys. **61**, 1077 (1993); doi: 10.1119/1.17354

[5] J Hull, *Options, Futures, and Other Derivatives*. Upper Saddle River, N.J: Pearson/Prentice Hall, 2006. Print.

[6] "The New York Stock Exchange." NYSE. N.p., n.d. Web. 5 Apr. 2016. <https://www.nyse.com/quote/index/!SPX>.

[7] Bart G. de Grooth, *A simple model for Brownian motion leading to the Langevin equation*, American Journal of Physics 67, 1248 (1999); doi: 10.1119/1.19111

[8] T Sauer, *Numerical Solutions to Stochastic Differential Equations in Finance*, George Mason University (Mathematics), Fairfax, VA

Appendix A: MATLAB Code

Part I: Generating Lambda Estimates

```
%% Estimating Lambda

load March21_FiveMinReturn_01
[steps_,num] = size(returnData);

Empirical_Data = returnData;
subplot(2,1,1)
hold on
title('Empirical Data Plot')
xlabel('Time (5min)')
ylabel('Return')
plot(Empirical_Data)
hold off

totalSteps = steps_;
totalSims = 1000;

num_points_lambdaInv = 100;
num_points_gamma = 10;
fin_lambdaInv = 12e-04; % On order of 1e-4
fin_gamma = 1.1;

lambdaInv = linspace(1e-04,fin_lambdaInv,num_points_lambdaInv);
gamma = linspace(0.9,fin_gamma,num_points_gamma);
stdDev = 1;

% Creates array stores steps
u = zeros(totalSteps,totalSims,num_points_gamma,num_points_lambdaInv);
% Generates all random steps; stores results in a large matrix
randomTerm =
normrnd(0,stdDev,totalSteps,totalSims,num_points_gamma,num_points_lambdaInv);

%% Simulation

for lambdaInv_Iter = 1:num_points_lambdaInv  % lambda
   for gamma_Iter = 1:num_points_gamma      % gamma
      for sim = 1:totalSims
         for iter = 1:totalSteps
            if iter==1
               step =
               (lambdaInv(lambdaInv_Iter))*randomTerm(iter,sim,gamma_Iter,lambdaInv_
               Iter);
```

```
            u(iter,sim,gamma_Iter,lambdaInv_Iter) = step;
        else
            step = -(gamma(gamma_Iter)*u(iter-
            1,sim,gamma_Iter,lambdaInv_Iter))+(lambdaInv(lambdaInv_Iter))*randomT
            erm(iter,sim,gamma_Iter,lambdaInv_Iter);
            u(iter,sim,gamma_Iter,lambdaInv_Iter) = u(iter-
            1,sim,gamma_Iter,lambdaInv_Iter) + step;
        end
      end
    end
  end
end

%% Combining results

results = zeros(totalSteps,num_points_gamma,num_points_lambdaInv);

for lambdaInv_Iter = 1:num_points_lambdaInv  % lambda
  for gamma_Iter = 1:num_points_gamma      % gamma
    for iter = 1:totalSteps
      total = sum(u(iter,:,gamma_Iter,lambdaInv_Iter));
      results(iter,gamma_Iter,lambdaInv_Iter) = total/totalSteps;
    end
  end
end

%% F Test for Variance

results_Std = zeros(num_points_gamma,num_points_lambdaInv);

for lambdaInv_Iter = 1:num_points_lambdaInv
  for gamma_Iter = 1:num_points_gamma
    tempTotal = 0;
    for sim = 1:totalSims
      temp = std(u(:,sim,gamma_Iter,lambdaInv_Iter));
      tempTotal = tempTotal + temp;
    end
    results_Std(gamma_Iter,lambdaInv_Iter) = tempTotal/totalSims;
  end
end

Ftest_Results = zeros(num_points_gamma,num_points_lambdaInv);
Empirical_Data_Std = std(Empirical_Data);

for lambdaInv_Iter = 1:num_points_lambdaInv
  for gamma_Iter = 1:num_points_gamma
```

```
        Ftest_Results(gamma_Iter,lambdaInv_Iter) =
         results_Std(gamma_Iter,lambdaInv_Iter)/Empirical_Data_Std;
    end
end

coefficient_Results_Ftest = zeros(1,3);
% note location (1,:) will contain zero since we started storing data at row 2)
% Range of results that will be caught by the IF statement
min_FScore = 0.9;
max_FScore = 1.1;

for lambdaInv_Iter = 1:num_points_lambdaInv
    for gamma_Iter = 1:num_points_gamma
        if (Ftest_Results(gamma_Iter,lambdaInv_Iter)>min_FScore) &&
            (Ftest_Results(gamma_Iter,lambdaInv_Iter)<max_FScore) &&
            lambdaInv(lambdaInv_Iter)~=0
            coefficient_Results_Ftest(end+1,1) = Ftest_Results(gamma_Iter,lambdaInv_Iter);
            coefficient_Results_Ftest(end,2) = gamma(gamma_Iter);
            coefficient_Results_Ftest(end,3) = lambdaInv(lambdaInv_Iter);
        end
    end
end

temp = coefficient_Results_Ftest;
coefficient_Results_Ftest = temp(2:end,:);

% Visualizing FTest results

subplot(2,1,2)
hold on
histogram(coefficient_Results_Ftest(:,3),30)
title('Langevin Model Results for {\lambda} fit')
xlabel('{\lambda}^-1 value')
ylabel('Frequency')
hold off

clc
disp('Program Complete')
```

Part 2: Generating Gamma Estimates

```
%% Estimating Lambda and Gamma

load March21_FiveMinReturn_01
[steps_,num] = size(returnData);
```

```
Empirical_Data = returnData;

totalSteps = steps_;
totalSims = 1000;

num_points_lambdaInv = 1;
num_points_gamma = 100;
fin_lambdaInv = 5.00e-04; % On order of 1e-4
fin_gamma = 1.25;

lambdaInv = fin_lambdaInv;
gamma = linspace(0,fin_gamma,num_points_gamma);
stdDev = 1;

% Creates array stores steps
u = zeros(totalSteps,totalSims,num_points_gamma,num_points_lambdaInv);
randomTerm =
normrnd(0,stdDev,totalSteps,totalSims,num_points_gamma,num_points_lambdaInv);

%% Simulation

for lambdaInv_Iter = 1:num_points_lambdaInv
   for gamma_Iter = 1:num_points_gamma
      for sim = 1:totalSims
         for iter = 1:totalSteps
            if iter==1
               step =
              (lambdaInv(lambdaInv_Iter))*randomTerm(iter,sim,gamma_Iter,lambdaInv_
              Iter);
               u(iter,sim,gamma_Iter,lambdaInv_Iter) = step;
            else
               step = -(gamma(gamma_Iter)*u(iter-
                1,sim,gamma_Iter,lambdaInv_Iter))+(lambdaInv(lambdaInv_Iter))*random
                Term(iter,sim,gamma_Iter,lambdaInv_Iter);
               u(iter,sim,gamma_Iter,lambdaInv_Iter) = u(iter-
                1,sim,gamma_Iter,lambdaInv_Iter) + step;
            end
         end
      end
   end
end

%% Time Spent Trending in a direction

results_Length = zeros(num_points_gamma,num_points_lambdaInv);
window = 10;
```

```
%Average trend length for empirical data
length = 0;
lengthStore = 0;
for i = 1:(steps_-1)
   if (Empirical_Data(i) * Empirical_Data(i+1)) < 0
      lengthStore(end+1) = length;
      length = 0;
   else
      length = length + 1;
   end
end

Empirical_Data_Length = mean(lengthStore(1:end));

for lambdaInv_Iter = 1:num_points_lambdaInv  % lambda
   for gamma_Iter = 1:num_points_gamma      % gamma
      meanLength_Sims = zeros(totalSims,1);
      for sims = 1:totalSims
         lengthStore_Sims = 0;
         lengthStore = 0;
         length = 0;
         for i = 1:(totalSteps-1)
            if (u(i,sim,gamma_Iter,lambdaInv_Iter) *
               u(i+1,sim,gamma_Iter,lambdaInv_Iter)) < 0
               lengthStore(end+1) = length;
               length = 0;
            else
               length = length + 1;
            end
         end
         meanLength_Sims(sims,1) = mean(lengthStore(1:end));
      end
      results_Length(gamma_Iter,lambdaInv_Iter) = mean(meanLength_Sims(:,1));
   end
end

coefficient_Results_Length = zeros(1,4);
% note location (1,:) will contain zero since we started storing data at row 2)
% Range of results that will be caught by the IF statement
min = 0.9;
max = 1.1;

for lambdaInv_Iter = 1:num_points_lambdaInv
   for gamma_Iter = 1:num_points_gamma
      if ((results_Length(gamma_Iter,lambdaInv_Iter)/Empirical_Data_Length)>min) &&
```

```
        ((results_Length(gamma_Iter,lambdaInv_Iter)/Empirical_Data_Length)<max)
        coefficient_Results_Length(end+1,1) =
        results_Length(gamma_Iter,lambdaInv_Iter);
        coefficient_Results_Length(end,2) =
        results_Length(gamma_Iter,lambdaInv_Iter)/Empirical_Data_Length;
        coefficient_Results_Length(end,3) = gamma(gamma_Iter);
        coefficient_Results_Length(end,4) = lambdaInv(lambdaInv_Iter);
      end
   end
end

% Eliminates zeros from the first row due to creation of matrix
temp = coefficient_Results_Length;
coefficient_Results_Length = temp(2:end,:);

clc
disp('Program Complete')
```

Part 3: Comparing Lambda and Gamma Estimates to Empirical Data

```
% Plotting Possible Lambda/Gamma Values

load March21_FiveMinReturn_01
[steps_,num] = size(returnData);

totalSteps = steps_;
totalSims = 1000;

lambdaInv = 5.00e-04;
gamma = 0.808;
stdDev = 1;

% Empirical Data
Empirical_Data = returnData;

% Creates array stores steps
u = zeros(totalSteps,totalSims);
randomTerm = normrnd(0,stdDev,totalSteps,totalSims);

%% Simulation

for sim = 1:totalSims
   for iter = 1:totalSteps
      if iter==1
         step = (lambdaInv)*randomTerm(iter,sim);
```

```
            u(iter,sim) = step;
        else
            step = -(gamma*u(iter-1,sim))+(lambdaInv)*randomTerm(iter,sim);
            u(iter,sim) = u(iter-1,sim) + step;
        end
    end
end

%% Standard Deviation at each time step

results = zeros(1,totalSteps);
for iter = 1:totalSteps
    results(iter) = std(u(iter,:));
end

%% Plotting Envelopes

subplot(2,1,1)
hold on
plot(Empirical_Data);
title('Empirical Market Data (2{\sigma} band)')
xlabel('Time Step (5min)')
ylabel('Return Value')
grid on
%legend('{\lambda}^-1=0.31');
plot(2*results(:),'r--','LineWidth',2);
plot(-2*results(:),'r--','LineWidth',2);
hold off

subplot(2,1,2)
hold on
for i = 1:1000:totalSims
    plot(u(:,i));
end
title('Langevin Model Results (2{\sigma} band)')
xlabel('Time Step (5min)')
ylabel('Return Value')
grid on
plot(2*results(:),'r--','LineWidth',2);
plot(-2*results(:),'r--','LineWidth',2);
hold off
```

Part 4: Estimating Alpha and Beta Values

```
% 3 Variable Model
```

```
load March23_FiveMinReturn_03
[steps_,num] = size(returnData);

Empirical_Data = returnData;

totalSteps = steps_;
totalSims = 100;

num_points_a = 20;
num_points_b = 20;
fin_a = 2e03;
fin_b =2e5;

%Constant
lambdaInv = 5.00e-04;
gamma = 0.808;

a = linspace(5e2,fin_a,num_points_a);
b = linspace(1e3,fin_b,num_points_b);

% Creates array stores steps    a          b
u = zeros(totalSteps,totalSims,num_points_a,num_points_b);

stdDev = 1;

%% Simulations

randomTerm = normrnd(0,stdDev,totalSteps,totalSims,num_points_a,num_points_b);
for b_Iter = 1:num_points_b          % b
   for a_Iter = 1:num_points_a   % a
      for sim = 1:totalSims
         for iter = 1:totalSteps
            clear historicalData
            clear historicalData_Squared
            historicalData = u(1:iter,sim,a_Iter,b_Iter);
            historicalData_Squared = u(1:iter,sim,a_Iter,b_Iter).^2;
            if iter <= 2
               uBar = 0;
               uSquared_Bar = 0;
               step = (lambdaInv)*randomTerm(iter,sim,a_Iter,b_Iter);
               u(iter,sim,a_Iter,b_Iter) =step;
            elseif iter <= 10 && iter > 2
               temp = tsmovavg(historicalData,'s',2,1);
               temp_Squared = tsmovavg(historicalData_Squared,'s',2,1);
               uBar = temp(iter);
               uSquared_Bar = temp_Squared(iter);
```

```
                step = -(gamma*u(iter-1,sim,a_Iter,b_Iter)) +
                (lambdaInv*((a(a_Iter)*uBar))) - (lambdaInv*b(b_Iter)*uSquared_Bar) +
                (lambdaInv)*randomTerm(iter,sim,a_Iter,b_Iter);
                u(iter,sim,a_Iter,b_Iter) = u(iter-1,sim,a_Iter,b_Iter) + step;
            else
                temp = tsmovavg(historicalData,'s',10,1);
                temp_Squared = tsmovavg(historicalData_Squared,'s',10,1);
                uBar = temp(iter);
                uSquared_Bar = temp_Squared(iter);
                step = -(gamma*u(iter-1,sim,a_Iter,b_Iter)) +
                (lambdaInv*((a(a_Iter)*uBar))) - (lambdaInv*b(b_Iter)*uSquared_Bar) +
                (lambdaInv)*randomTerm(iter,sim,a_Iter,b_Iter);
                u(iter,sim,a_Iter,b_Iter) = u(iter-1,sim,a_Iter,b_Iter) + step;
            end
        end
    end
end

%% Testing using Geomean

% Geomean of Empirical Data
Geomean_Empirical_Data = geomean(Empirical_Data + 1)-1;
Geomean_results = zeros(num_points_a,num_points_b);

% Prepare u data (removing NaN's and +/-inf, replacing whole column
% with large number (10) that can be identified as erroneous)
u_adjusted = u;

for b_Iter = 1:num_points_b      % b
    for a_Iter = 1:num_points_a  % a
        for sim = 1:totalSims
            if isnan(u(end,sim,a_Iter,b_Iter)) || u(end,sim,a_Iter,b_Iter) == inf ||
                u(end,sim,a_Iter,b_Iter) == -inf || u(end,sim,a_Iter,b_Iter) > 1 ||
                u(end,sim,a_Iter,b_Iter) < -1
                for i = 1:totalSteps
                    u_adjusted(i,sim,a_Iter,b_Iter) = 100;
                end
            end
        end
    end
end

for b_Iter = 1:num_points_b
    for a_Iter = 1:num_points_a
        total = 0;
```

```
        for sim = 1:totalSims
            sample = geomean(u_adjusted(:,sim,a_Iter,b_Iter) + 1) - 1;
            total = total + sample;
        end
        Geomean_results(a_Iter,b_Iter) = total/totalSims;
    end
end

coefficient_Geomean_Results = zeros(1,4);   % note location (1,:) will contain zero since
we started storing data at row 2)
% Range of results that will be caught by the IF statement
min = 1.2;%-1e-5;
max = 0.8;%1e-5;

for b_Iter = 1:num_points_b
    for a_Iter = 1:num_points_a
        if (abs(Geomean_results(a_Iter,b_Iter)/Geomean_Empirical_Data)>min) &&
            (abs(Geomean_results(a_Iter,b_Iter)/Geomean_Empirical_Data)<max)
            coefficient_Geomean_Results(end+1,1) = Geomean_results(a_Iter,b_Iter);
            coefficient_Geomean_Results(end,2) =
            Geomean_results(a_Iter,b_Iter)/Geomean_Empirical_Data;
            coefficient_Geomean_Results(end,3) = a(a_Iter);
            coefficient_Geomean_Results(end,4) = b(b_Iter);
        end
    end
end

% Eliminates zeros from the first row due to creation of matrix
temp = coefficient_Geomean_Results;
coefficient_Geomean_Results = temp(2:end,:);

clc
disp('Program Complete')
```

Part 5: Plotting Full Langevin Equation vs. Empirical Data

```
% Three Coefficient Model (Constants)

load March21_FiveMinReturn_01
[steps_,num] = size(returnData);

Empirical_Data = returnData;

totalSteps = steps_;
totalSims = 1000;
```

```
%Constant
a =   973.6842;
b = 1.2668e+05;
lambdaInv = 5.00e-04;
gamma = 0.808;

% Creates array stores steps
u = zeros(totalSteps,totalSims);

stdDev = 1;

%% Simulations

randomTerm = normrnd(0,stdDev,totalSteps,totalSims);
for sim = 1:totalSims
   for iter = 1:totalSteps
      clear historicalData
      clear historicalData_Squared
      historicalData = u(1:iter,sim);
      historicalData_Squared = u(1:iter,sim).^2;
      if iter <= 2
         uBar = 0;
         uSquared_Bar = 0;
         step =(lambdaInv*b*uSquared_Bar) + (lambdaInv)*randomTerm(iter,sim);
         u(iter,sim) = step;
      elseif iter <= 10 && iter > 2
         temp = tsmovavg(historicalData,'s',2,1);
         temp_Squared = tsmovavg(historicalData_Squared,'s',2,1);
         uBar = temp(iter);
         uSquared_Bar = temp_Squared(iter);
         step = -(gamma*u(iter-1,sim)) + (lambdaInv*((a*uBar))) -
         (lambdaInv*b*uSquared_Bar) + (lambdaInv)*randomTerm(iter,sim);
         u(iter,sim) = u(iter-1,sim) + step;
      else
         temp = tsmovavg(historicalData,'s',10,1);
         temp_Squared = tsmovavg(historicalData_Squared,'s',10,1);
         uBar = temp(iter);
         uSquared_Bar = temp_Squared(iter);
         step = -(gamma*u(iter-1,sim)) + (lambdaInv*((a*uBar))) -
         (lambdaInv*b*uSquared_Bar) + (lambdaInv)*randomTerm(iter,sim);
         u(iter,sim) = u(iter-1,sim) + step;
      end
   end
end

% Graphing some results
```

```
subplot(2,1,1)
hold on
plot(Empirical_Data)
for i = 1:100:totalSims
   plot(u(:,i))
end
grid on
hold off

%% Standard Deviation at each time step

results = zeros(1,totalSteps);
for iter = 1:totalSteps
   results(iter) = std(u(iter,:));
end

%% Plotting Envelopes

subplot(2,1,2)
hold on
plot(Empirical_Data);
title('Empirical Market Data (2{\sigma} band)')
xlabel('Time Step (5min)')
ylabel('Return Value')
grid on
%legend('{\lambda}^-1=0.31');
plot(2*results(:),'r--','LineWidth',2);
plot(-2*results(:),'r--','LineWidth',2);
hold off
```