Factors in Interest Rate Models

by

Gregory Morneault

Certificate in Quantitative Finance

January 2015

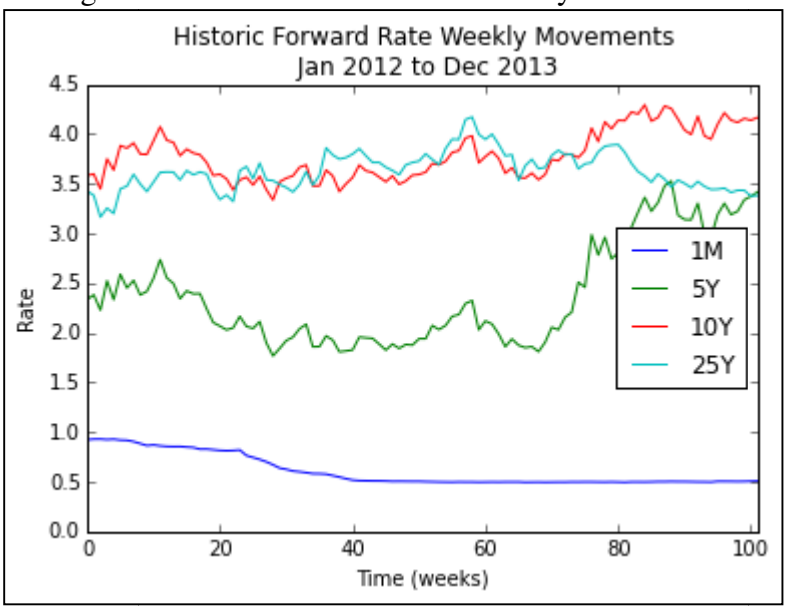
Table of Contents

DATA

Forward rates data for the Pound Sterling pulled from the Bank of England site. For the purposes of pricing caps and floors and zero coupon bonds, the Commercial Bank Liability Curve (BLC), considered to be a LIBOR equivalent, is used.

Data from January 2012 to December 2013 was used in the calibration of the model. A weekly frequency (data points taken from the Wednesday of each week) was used to limit the amount of noise in the movements of the rates – especially the short term part of the forward rate curve. Since the sample is pulled from a low rate regime, even slight movements in the short term forward rate can have a significant impact on PCA results.

Figure 1: Historic Forward Rate Weekly Movements



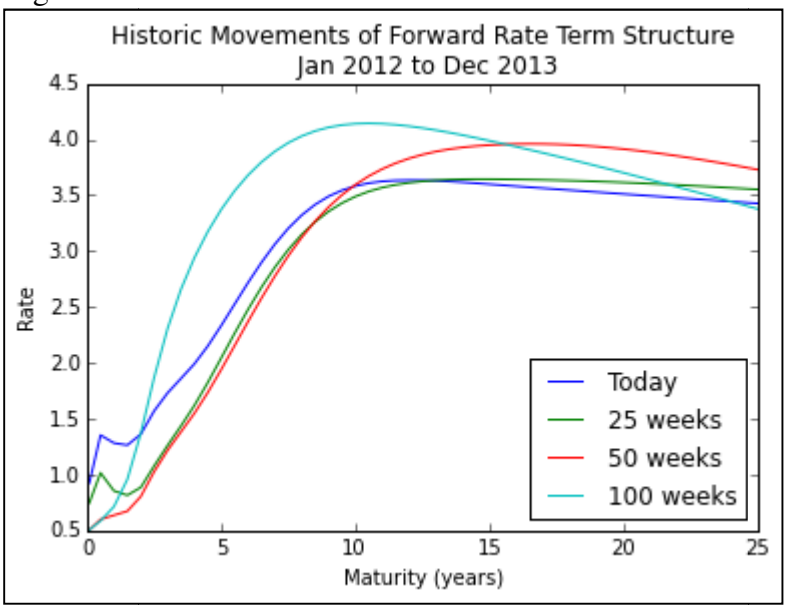
An examination of the historical weekly forward rates shows that there was an initial decrease and subsequent flattening of the 1 month forward rate. The 5 and 10 year rates are highly correlated while the 1 month and 25 year rates demonstrate some degree of negative correlation.

Table 1: Historic Forward Rate Correlation Matrix between Jan 2012 and Dec 2013

	1M	5Y	10Y	25Y
1M	1.000	-0.107	-0.238	-0.513
5Y	-0.107	1.000	0.927	-0.302
10Y	-0.238	0.927	1.000	-0.003
25Y	-0.513	-0.302	-0.003	1.000

The moment of the yield curve remained fairly stable for 2012. In 2013 the short term end of the curve began to steepen while long-term rates began to demonstrate some degree of inversion.

Figure 2: Historic Forward Rate Term Structure Movements



FACTORIZATION AND CALIBRATION

PRINCIPAL COMPONENT ANALYSIS

The BOE data contains instantaneous forward rates up to 25 years in half-year increments. Principal component analysis (PCA) is used to determine dominant and independent factors that can adequately describe the movements of each of the forward rates over the two year sample.

To conduct PCA on the sample, a covariance matrix was calculated on the log-differences in forward rates over the two year time span of the sample data. Log-differences were used because January 2012 to December 2013 is in a low rate regime.

Covariance was calculated using the standard NumPy covariance implementation[1] from the widely known LAPACK[2] (Linear Algebra PACKage) library. The resulting matrix was re-annualized by a factor of $\frac{51}{100 * 100}$ to account for the fact that the data was weekly and the rates were displayed as percentages (i.e. 0.05 as 5). The numerator is 51 as opposed to 52 to account for bank holidays throughout the year.

The standard NumPy eigenvalue function[3] which leverages LAPACK was used to calculate the eigenvalues and eigenvectors on the covariance matrix. The eigenvalues were then placed in descending order and the cumulative R^2 values were calculated thusly: $Cum.R^2 = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{N} \lambda_i}$. A separate method using QR decomposition[4] was used to verify the validity and accuracy of the eigenvalues and eigenvectors. Both implementations had nearly identical eigenvalues and the table below shows the cumulative R^2 values for both the NumPy and QR decomposition methods had the same ranking and explanatory power.

[1] http://docs.scipy.org/doc/numpy/reference/generated/numpy.cov.html
[2] http://www.netlib.org/lapack/
[3] http://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.eig.html
[4] Note that the QR decomposition implementation was leveraged purely for validating the results of the NumPy implementation and was not used in calculating the volatility and drift functions discussed later in the paper.

Table 2: Cumulative R2 for Numpy and QR Decomposition

Eigenvector (sorted)	Numpy Cum. R2	QR Decomposition Cum. R2
e(1)	0.678	0.678
e(2)	0.865	0.865
e(3)	0.942	0.942
e(4)	0.964	0.964
e(5)	0.980	0.980
e(6)	0.990	0.990
e(7)	0.995	0.995
e(8)	0.997	0.997
e(9)	0.999	0.999
e(10)	1.000	1.000
⋮	⋮	⋮
e(51)	1.000	1.000

It is suggested to have a sufficient number of factors to cover at least 95% of the movement of the curve. The first for principal components accomplish this with a cumulative R^2 value of 96.4%. The table below illustrates the shape of the four eigenvectors for both NumPy and QR decomposition methods.

Figure 3: First Four Factors – Numpy

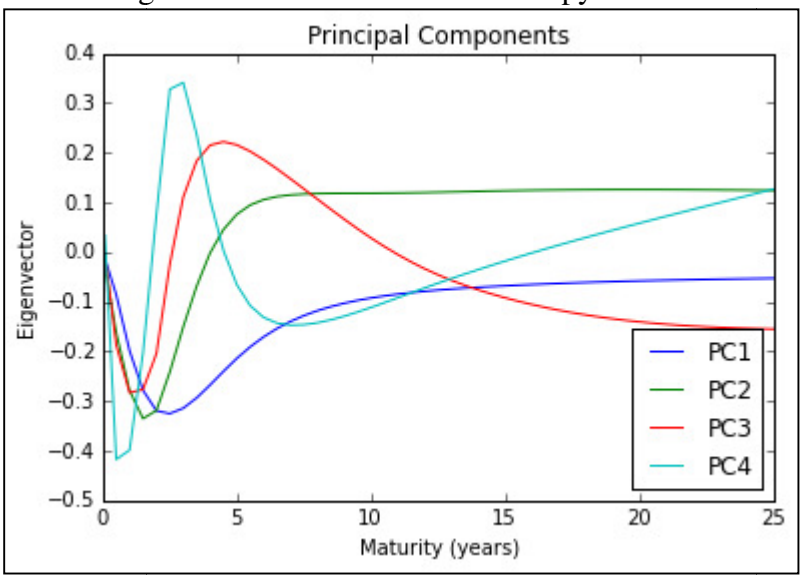
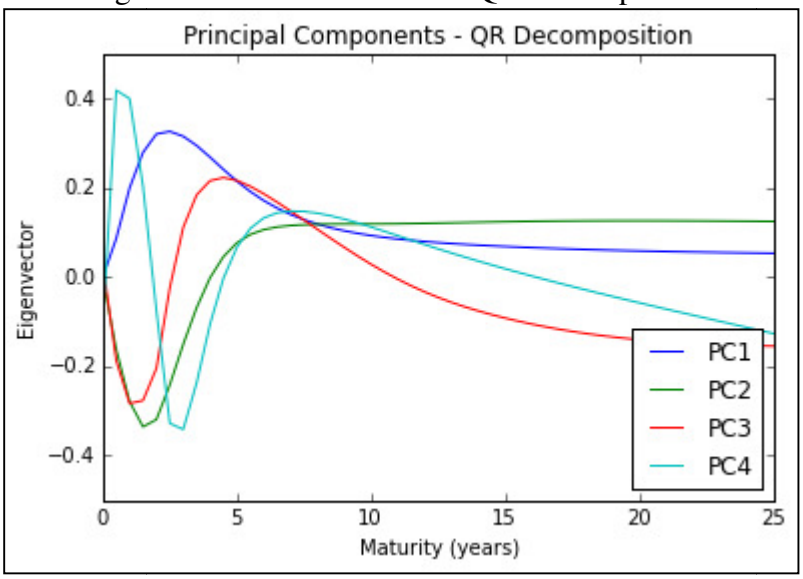


Figure 4: First Four Factors – QR Decompostion



While the factors exhibit the same shape for both implementations, the signs for the first and fourth eigenvectors are flipped. This is still acceptable since the factors maintain their orthogonality and is likely a result of the differences in initialization of the two algorithms.

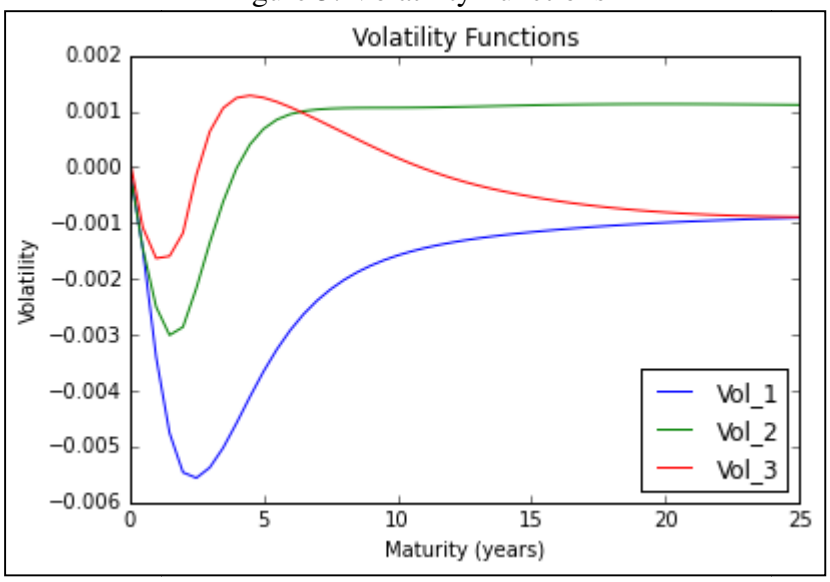
The first principal component never crosses zero and so can be considered to represent the parallel shift in the curve. The second principal component crosses zero at around the 5 year maturity and can be seen to represent the skew behavior of the curve. The third principal component crosses zero at the 3 year maturity and again at the 10 year maturity and represents a twisting of the curve. The fourth principal component also illustrates a twisting behavior in the curve.

VOLATILITY FUNCTIONS

Musiela parameterization of the HJM SDE volatility functions is calculated as follows: $\bar{v}_i(t,\tau) = \sqrt{\lambda_i}\mathbf{e}^{(i)}$. After transforming the top four principal components into volatility functions, an attempt was made to fit a polynomial to the fourth volatility

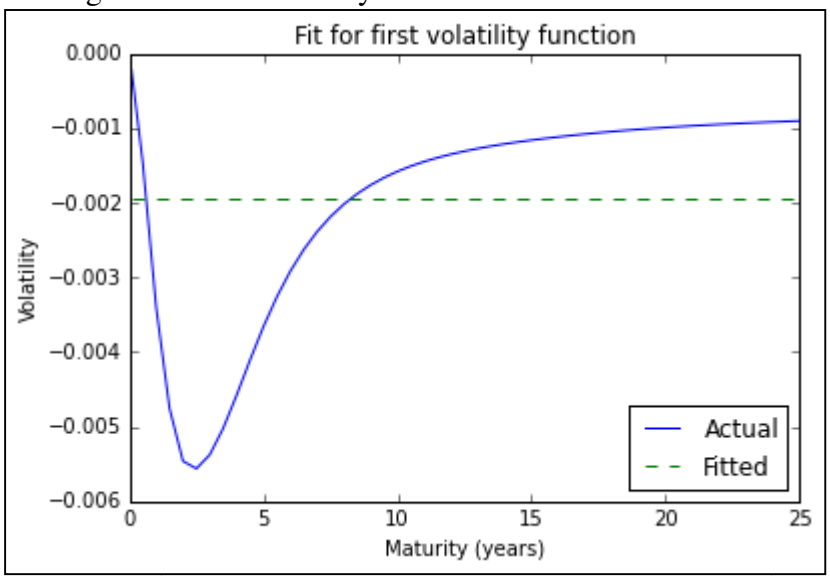
function. However, due to the majority of the twisting being represented in the short rate part the curve, a good fit of the curve proved difficult. Since even without the fourth volatility function, the first three factors still accounted for 94.2% coverage of the curve moment, it was determined to be acceptable to drop the fourth volatility function. The first three volatility functions are shown in the figure below.

Figure 5: Volatility Functions



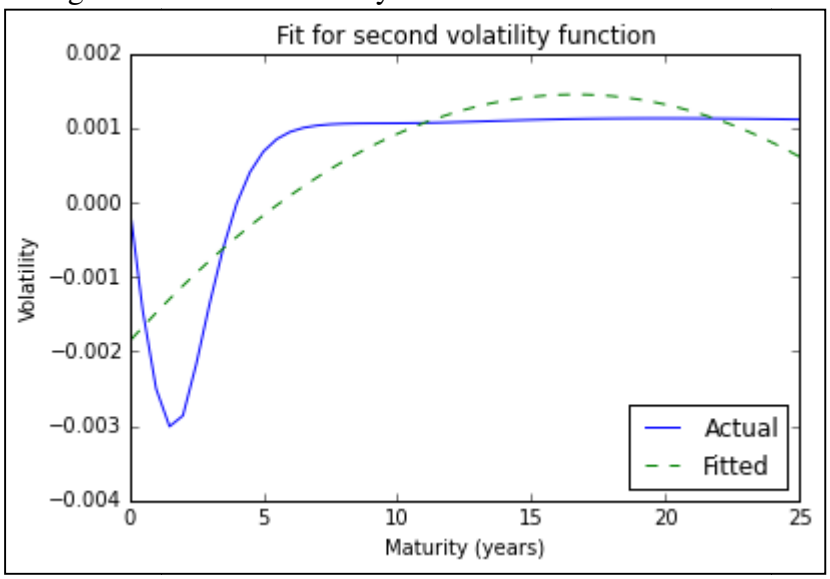
The first volatility function was fitted with a zero degree polynomial since it is representative of a parallel shift in the curve. While there is a noticeable hump at the 3 year tenor it was determined that a linear or even second degree polynomial would not be appropriate since the volatility function does not change sign at any point.

Figure 6: First Volatility Function Actual vs. Fit



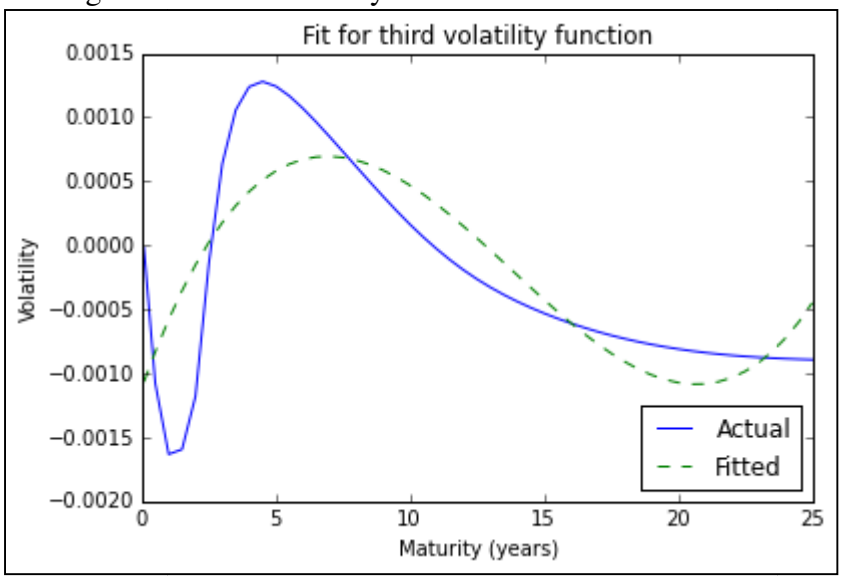
The second volatility function was fit with a two degree polynomial. While a third degree polynomial did provide a better fit, limiting the fit to a second degree polynomial was deemed sufficient since the volatility function only changed sign once and the fit was still able to account for the skew in the curve between the short and long term sections of the curve.

Figure 7: Second Volatility Function Actual vs. Fit



The third volatility function was fit with a third degree polynomial.

Figure 8: Third Volatility Function Actual vs. Fit



DRIFT FUNCTION

One of the important results of HJM model is that the risk-neutral drift must satisfy the no-arbitrage restriction:

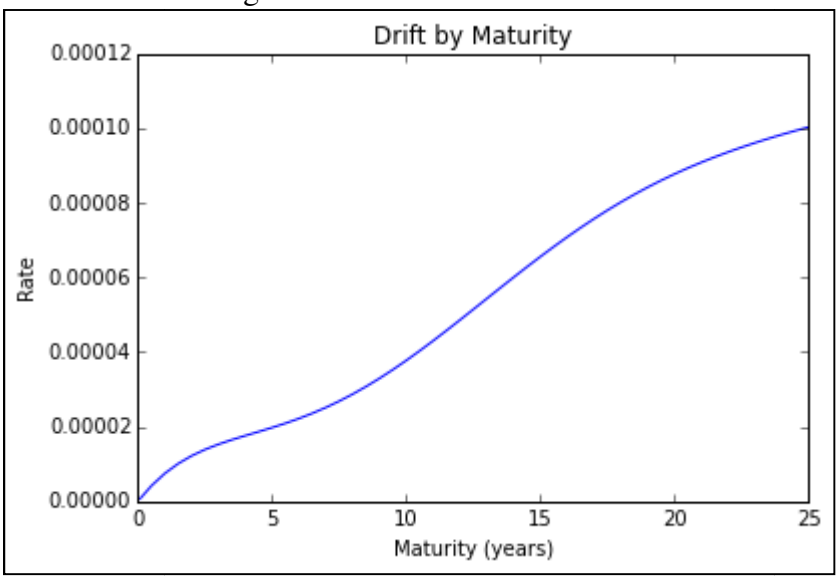
$$m(t,T) = \sum_{i=1}^{n} v_i(t,T) \int_t^T v_i(t,s)ds$$

Under Musiela parameterization $\bar{v}(t,\tau) = v(t,T)$ so that:

$$\bar{m}(t,T) = \sum_{i=1}^{n} \bar{v}_i(t,\tau) \int_0^\tau \bar{v}_i(t,s)ds$$

To calculate the drift function we use numerical integration over the fitted volatility functions. The trapezium rule was used for integration purposes and a value of $d\tau = 0.01$ was deemed to be sufficient for the purposes of estimating the drift function. The table below shows the monotonicity of the drift function which is an important characteristic of the result.

Figure 9: Drift Function



RANDOM NUMBER GENERATION

Two separate random number generators were used in the implementation of the Monte Carlo simulations: a pseudo-random NumPy implementation[5] of the Mersenne Twister and a low-discrepancy Sobol implementation[6] courtesy of John Burkart of Florida State University. Throughout the remainder of the paper these two implementations will be referred to as Numpy and Sobol respectively.

Both methodologies generated random numbers uniformly distributed between zero and one. A well-known method of generating normally distributed random numbers is to convert 12 independently distributed uniform random numbers into a single normally distributed random number as follows: $Z = \sum_{i=1}^{12} U_i - 6$. For Sobol, special consideration had to be given to deal with dimensionality. The algorithm allows for the

[5] http://docs.scipy.org/doc/numpy/reference/generated/numpy.random.uniform.html
[6] http://people.sc.fsu.edu/~jburkardt/py_src/sobol/sobol.html

setting the dimension (in this case 12) so that the observations can be considered "independent" for the purposes of generating a normally distributed random number.

An experiment was performed where the average mean and standard deviation for both implementations was taken from 100 iterations of a 1,000 sample dataset. The Sobol implementation was more representative of a normal distribution with an average mean and standard deviation closer to 0 and 1 respectively.

Figure 10: Numpy Average Mean and Std Dev

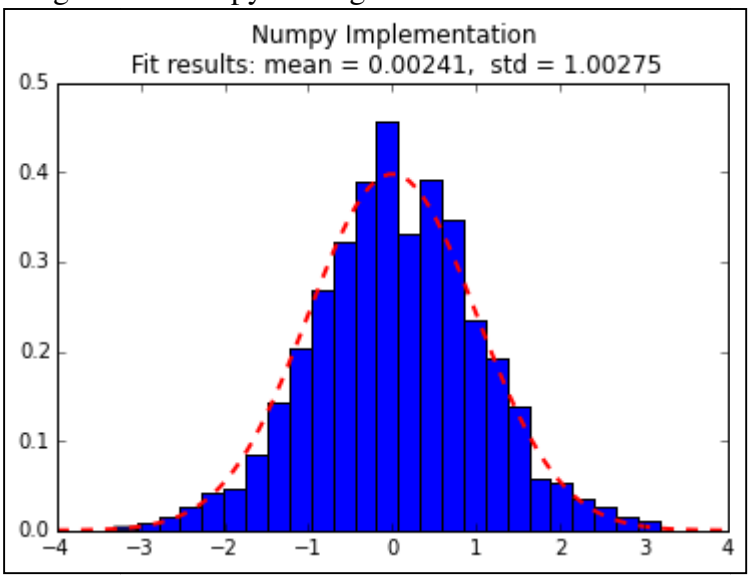
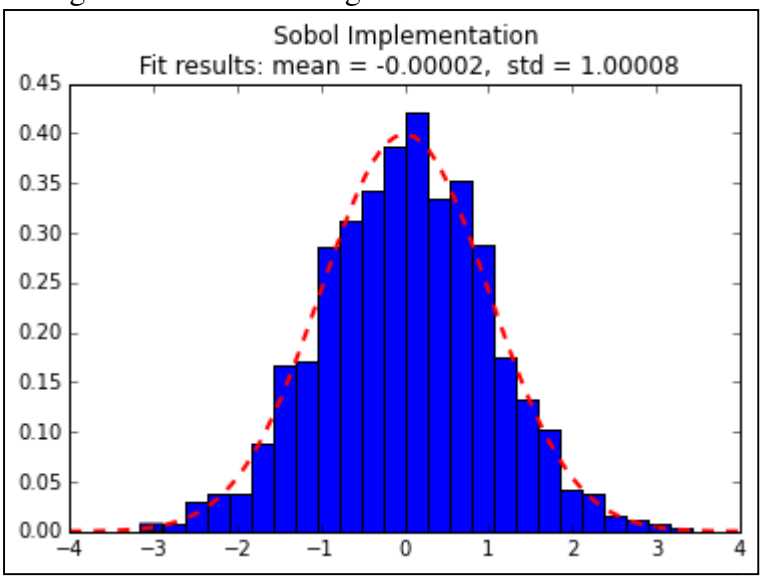


Figure 11: Sobol Average Mean and Std Dev



Although Sobol was more accurate, the average processing time to generate the 1,000 number sample was significantly longer at 0.06 seconds compared to a nearly instantaneous return for Numpy. While it is expected that Sobol should have a faster run time since it is a low-discrepancy sequence and Numpy is pseudo-random, the fact that Numpy was implemented in lower level C language more than offset this supposed loss in efficiency.

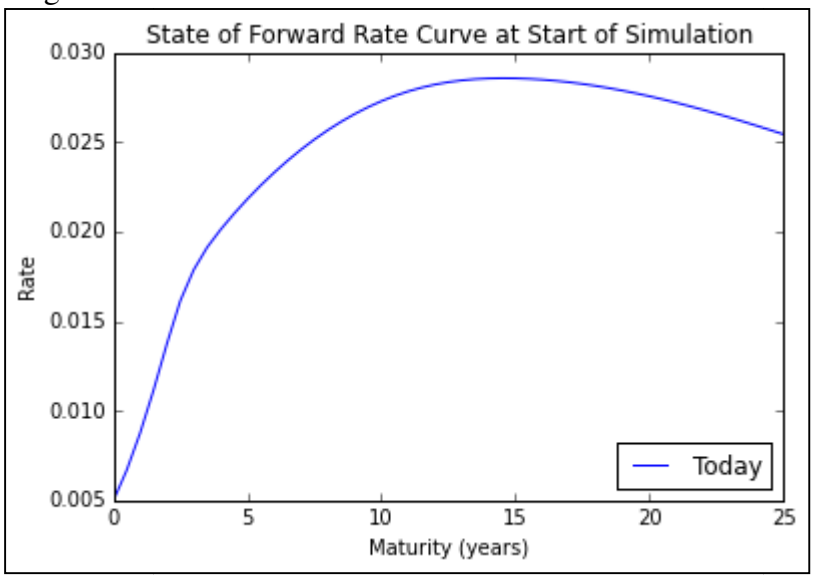
It is also worth noting that the Sobol implementation takes significantly longer to run when the seed is not incremental from one execution to the next. The algorithm stores the global seed value after each execution, if the seed is not "next in line" then the algorithm needs to "reset". In the implementation of the Monte Carlo that follows, the seed is tracked and incremented accordingly to allow for faster computation.

FORWARD RATE CONSTRUCTION

At the time of conducting this simulation, the most recent forward rate curve data available from the Bank of England site was November 28th, 2014 and is chosen to be the

starting point of all HJM simulations. The figure below illustrates the shape of the curve at this point in time.

Figure 12: Forward Rate Curve at Start of Simulation



With Musiela parameterization, the SDE to implement the multi-factor HJM model becomes:

$$d\bar{f}(t,\tau) = \bar{m}(t,\tau) + \sum_{i=1}^{N} \bar{v}_i(t,\tau)dX_i + \frac{\partial \bar{f}(t,\tau)}{\partial \tau}$$

Where the forward rate derivative, $\frac{\partial \bar{f}(t,\tau)}{\partial \tau}$, is calculated using the previous row of the simulation.

The figures below illustrate a single simulation for both the Numpy and Sobol implementations.

Figure 13: Numpy Projection of Forward Rates

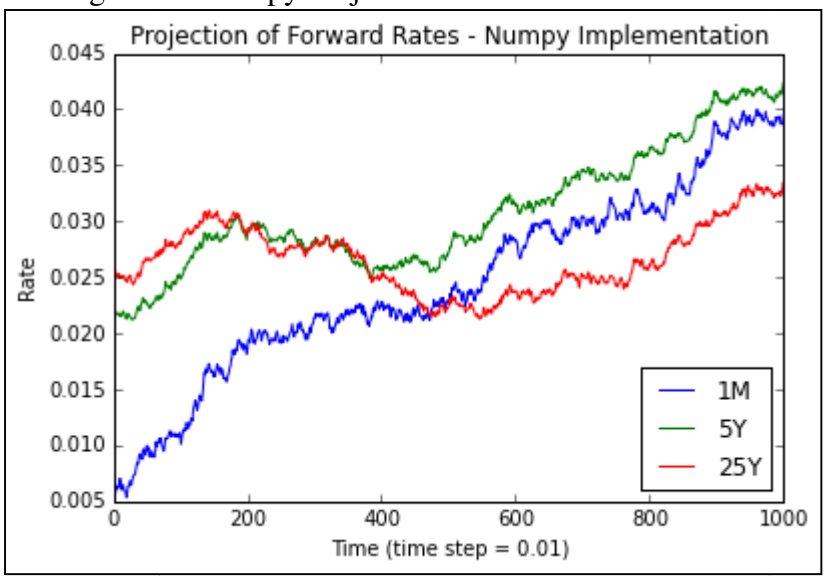


Figure 14: Sobol Projection of Forward Rates

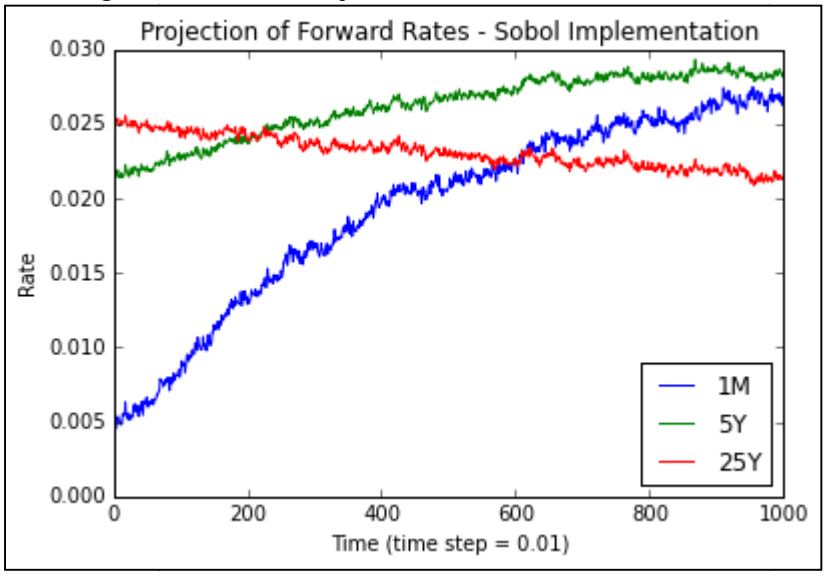


Figure 15: Numpy Evolution of Forward Rate Curve

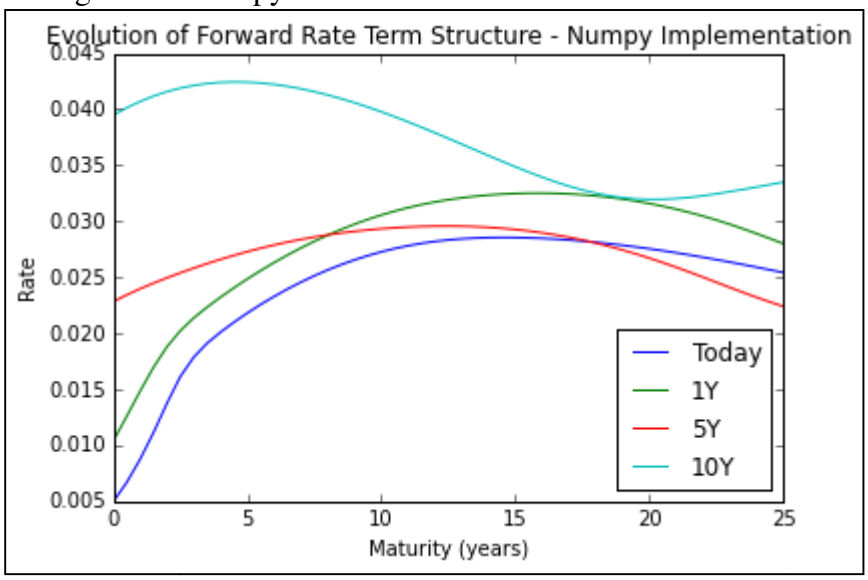
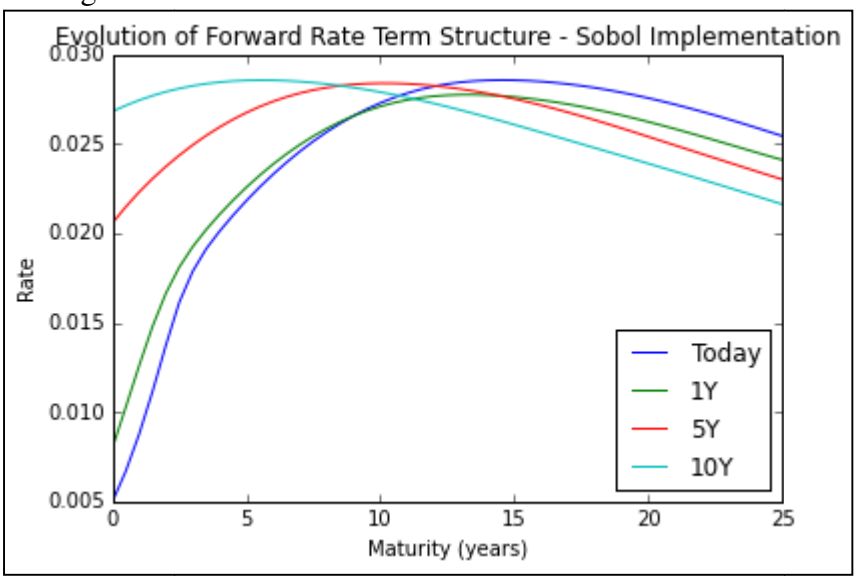


Figure 16: Sobol Evolution of Forward Rate Curve



It is apparent from just one simulation that the Sobol implementation exhibits much more stability in the forward rate projections and subsequently in the forward rate curve as a whole. After conducting visual examination of numerous simulations, Sobol seemed to exhibit very little variation in the simulated movement of the forward rate curve while Numpy had more varied results. Both Numpy and Sobol tended to evolve

toward an inverted forward rate curve toward the end of the simulation (at 10 years). There may likely be multiple contributing factors as to the cause of this outcome.

First, the starting state of the forward curve was very steep at the short term rates. It is likely that the forward rate derivative, $\frac{\partial \bar{f}(t,\tau)}{\partial \tau}$, contributed to a "pulling" effect where the derivative has the largest impact for each time step since the rate difference between the tenors were greatest. For the same reason, it is likely that the longer rates faced a slight downward pull since that part of the curve had started inverted.

Second, the second volatility function which represents the skew may have had a large impact on the tendency toward inversion. Volatilities were greatest in magnitude for the shorter maturities and slightly negatively correlated to the longer maturities.

MONTE CARLO

CONVERGENCE

A 10,000 simulation Monte Carlo was executed for both the Numpy and Sobol implementations. For comparison purposes and to demonstrate rate of conversion, the average value of a caplet with a 6 month expiry and 1year maturity with a strike of 150 bps was plotted over 500, 2,000, and 10,000 simulations.

At 500 simulations it is already quite clear that Sobol converged rather quickly. The Numpy implementation, on the other hand, continues to show significant variation until around 3,000 simulations and doesn't begin to level out until well after 6,000 simulations.

Figure 17: Numpy at 500 Simulations

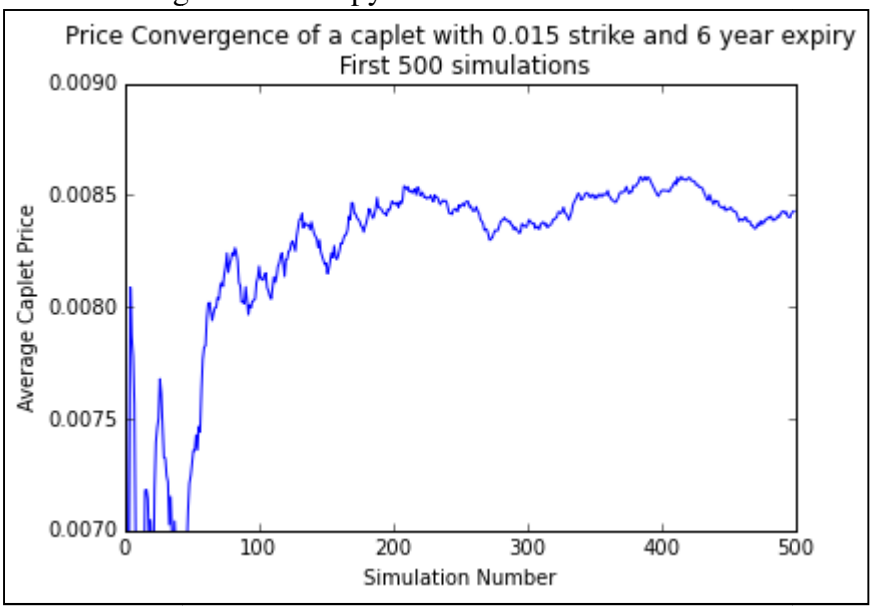


Figure 18: Sobol at 500 Simulations

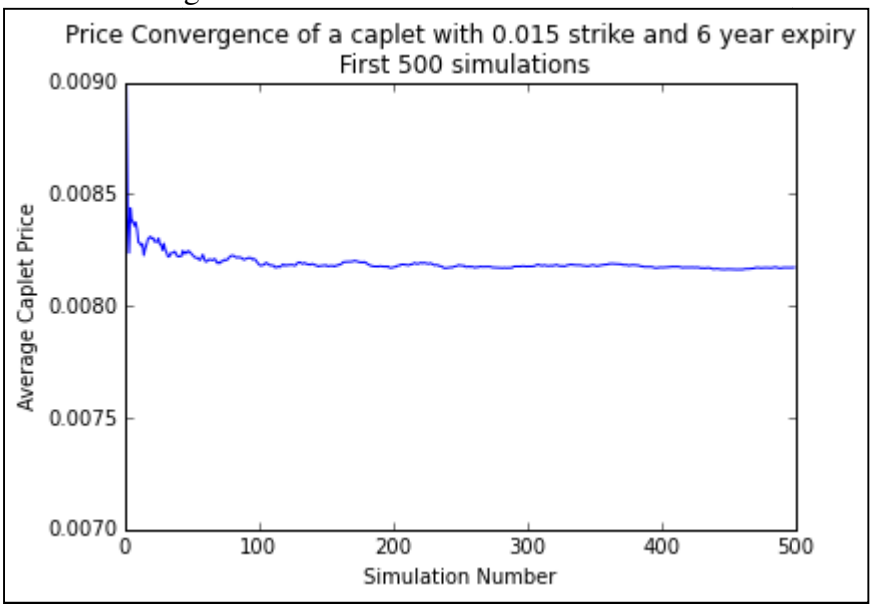


Figure 19: Numpy at 2,000 Simulations

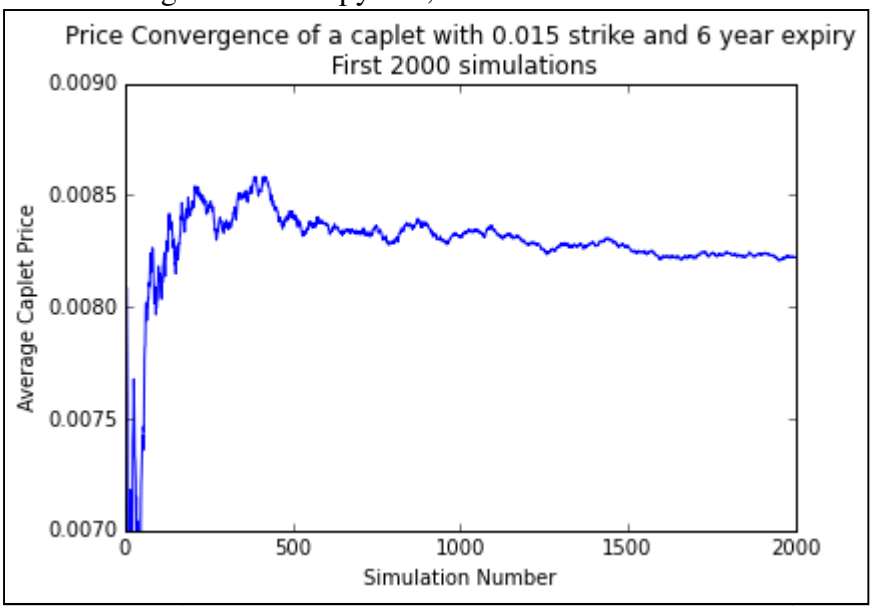


Figure 20: Sobol at 2,000 Simulations

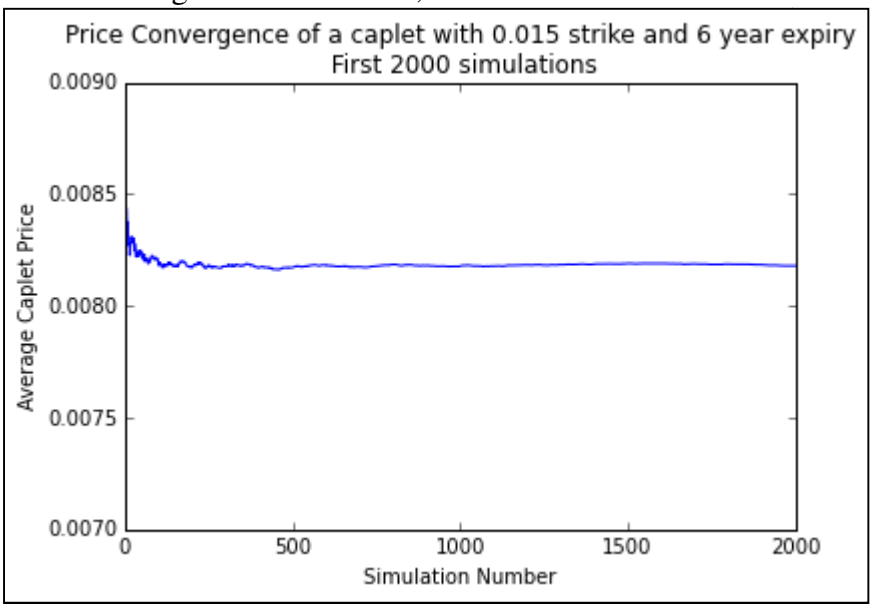


Figure 21: Numpy at 10,000 Simulations

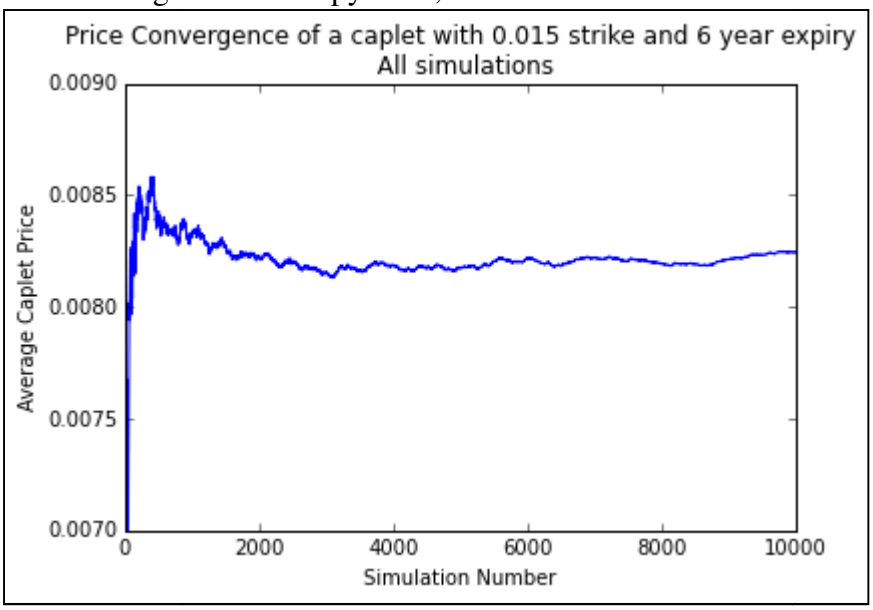
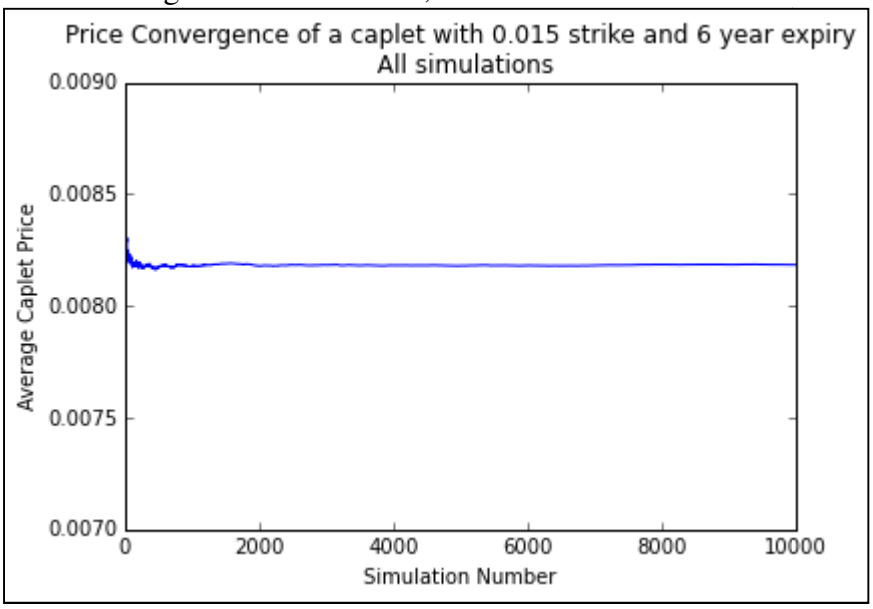


Figure 22: Sobol at 10,000 Simulations



COMPUTATION TIME

The Sobol implementation proved to be approximately 50% slower as compared to the Numpy simulation. However, based on the fast convergence rate of Sobol, it is likely that Sobol would take fewer simulations, and consequentially less computation time, to reach a given convergence threshold.

Figure 3: Processing Time for 10,000 Simulation Monte Carlo (seconds)

	All Simulations	Per Simulation
Numpy	2,408	0.24
Sobol	3,779	0.38
Total	6,186	0.62

Again, it is worth noting that the Sobol implementation was conducted in the Python language whereas the Numpy implementation was coded in the lower and more computationally efficient language, C, and wrapped in Python for ease of use. Assuming both implementations had used the same language, Sobol would likely have had a faster computation time while still retaining a faster rate of convergence.

PRICING

CAPS AND FLOORS

Pricing was conducted for caplets, floorlets across a range of expiries and strikes on the 1 month rate. All caplets and floorlets had a maturity of 1 year following expiration and prices were calculated by taking average values across all simulations. The figures below plot caplet and floorlet prices by strike (or "moneyness") for each of 2, 5, and 10 year expiries. The short term rate as of the time of the pricing was 0.52% (0.005 on the scale below).

Figure 23: Numpy Caplet Pricing at various Strikes

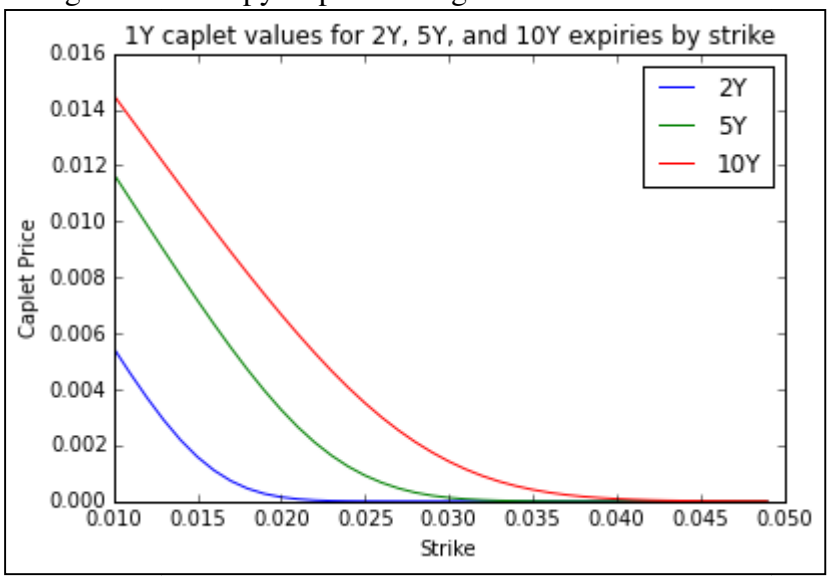


Figure 24: Numpy Floorlet Pricing at various Strikes

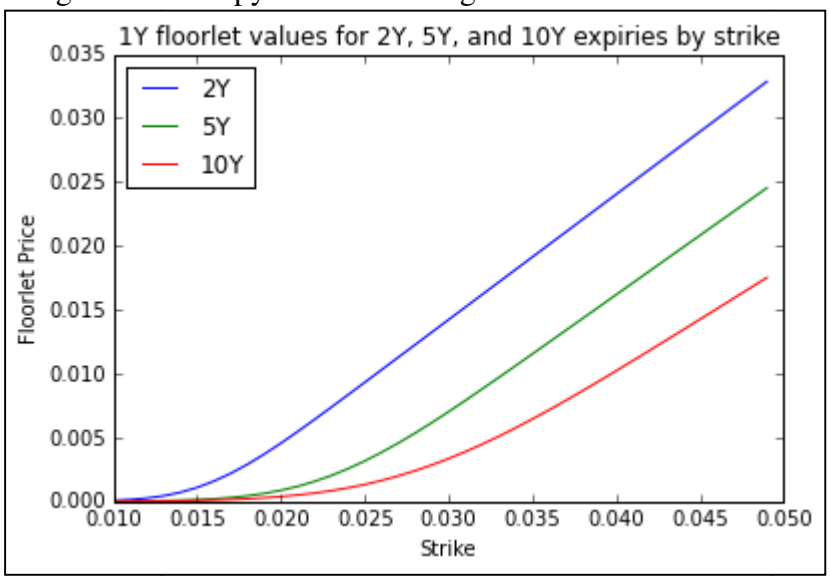


Figure 25: Sobol Caplet Pricing by Strike

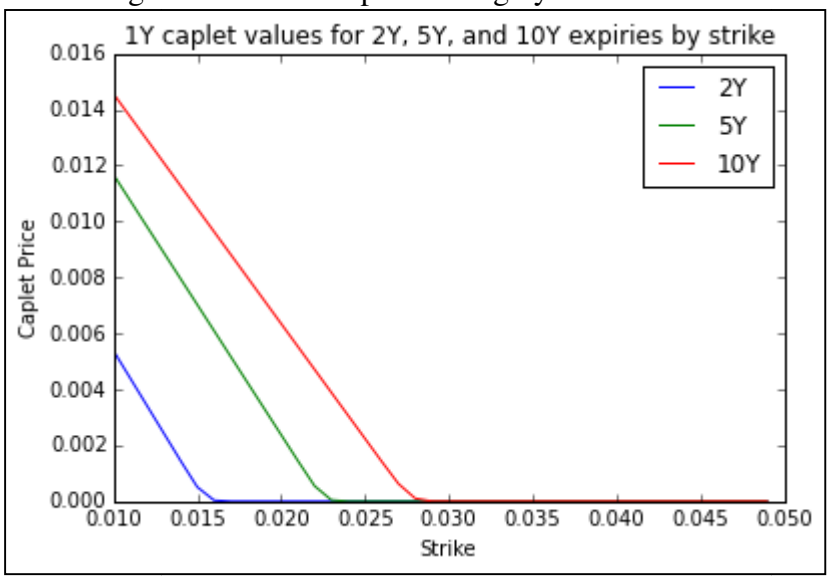
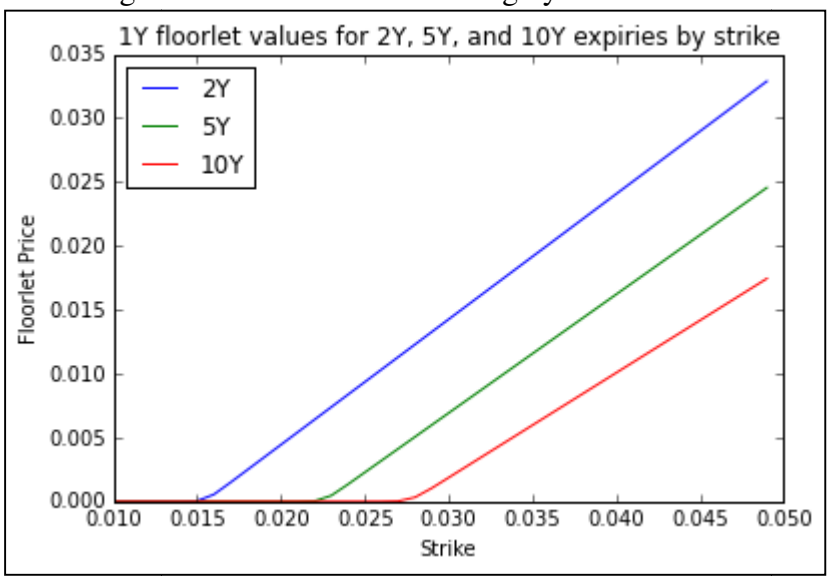


Figure 26: Sobol Floorlet Pricing by Strike



The payoff diagrams clearly depict that Sobol is more consistent at arriving at a price as is apparent by the rapid change in delta where the simulated rate crosses the strike. In other words, it is easy to see from the Sobol payoff diagrams that the simulated 1 month forward rate is consistently at 1.5% at year two, 2.25% at year five, and 2.75% at year ten as these are the point at which the delta rapidly changes from zero to one.

Numpy has more gradual change in delta due to the greater degree of variance over all simulations.

It can also be observed that there is clearly a gradual trend for the 1 month forward rate to increase overtime for both simulations. This is evident from the "shifting" of the payoff diagrams from lower strikes to higher strikes with an increase in expiry. As mentioned earlier, there is likely more than one contributing factor leading to the increase in short term forward rates over the simulation period. Namely, the impact that the forward rate derivative has on a steep rate curve and the impact of the second volatility function representing the skew.

IMPLIED VOLATILITY

Implied volatilities were calculated on the Black formula using the bisection method. The figures below show the volatility surfaces for the same caplets and floorlets by strike and maturity.

Figure 27: Numpy Caplet Volatility Surface

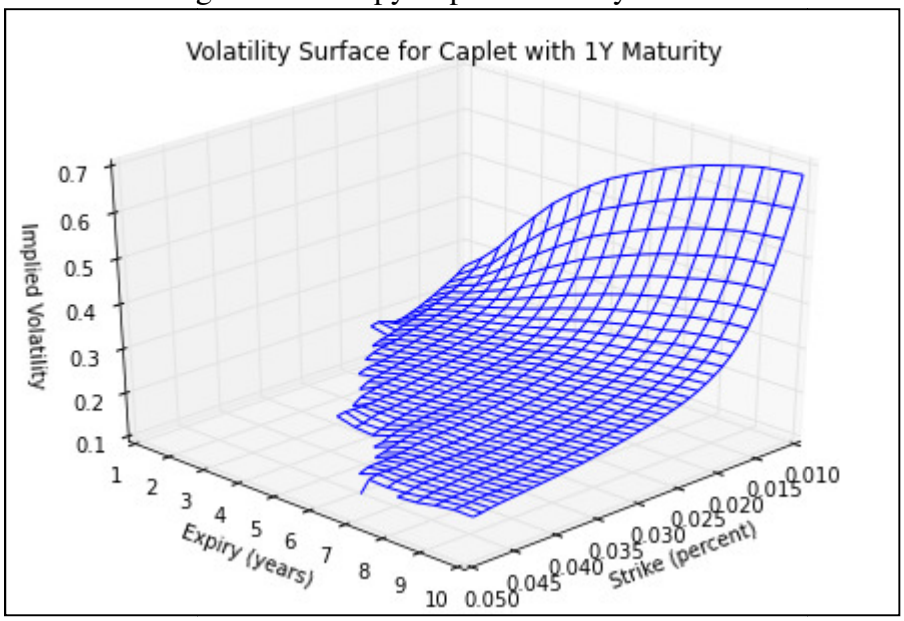


Figure 28: Sobol Caplet Volatility Surface

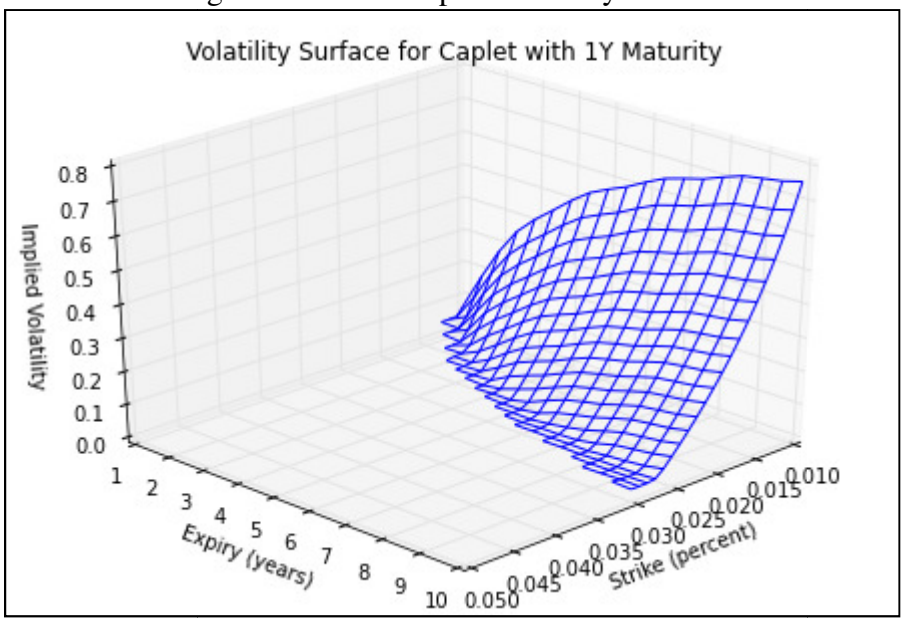


Figure 29: Numpy Floorlet Volatility Surface

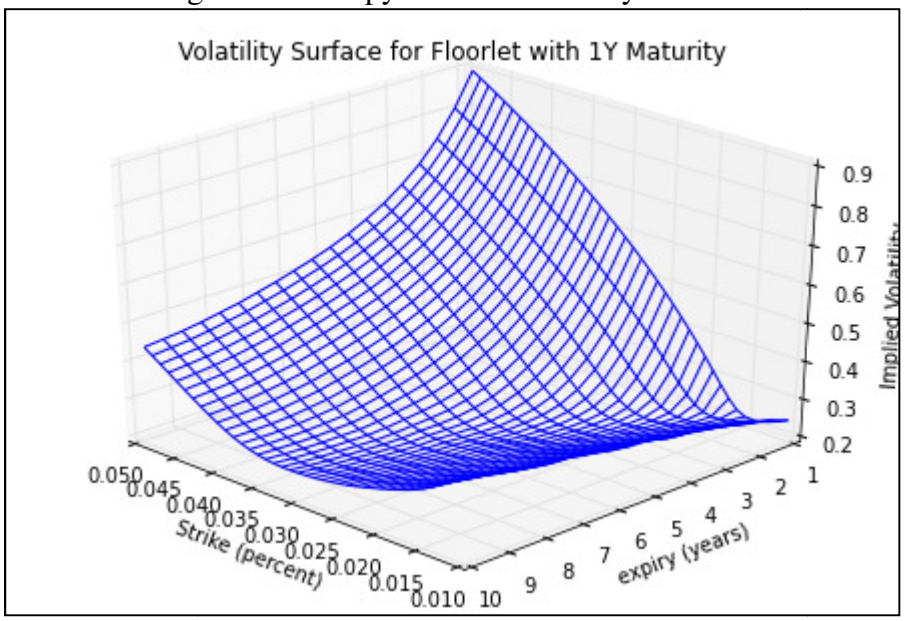
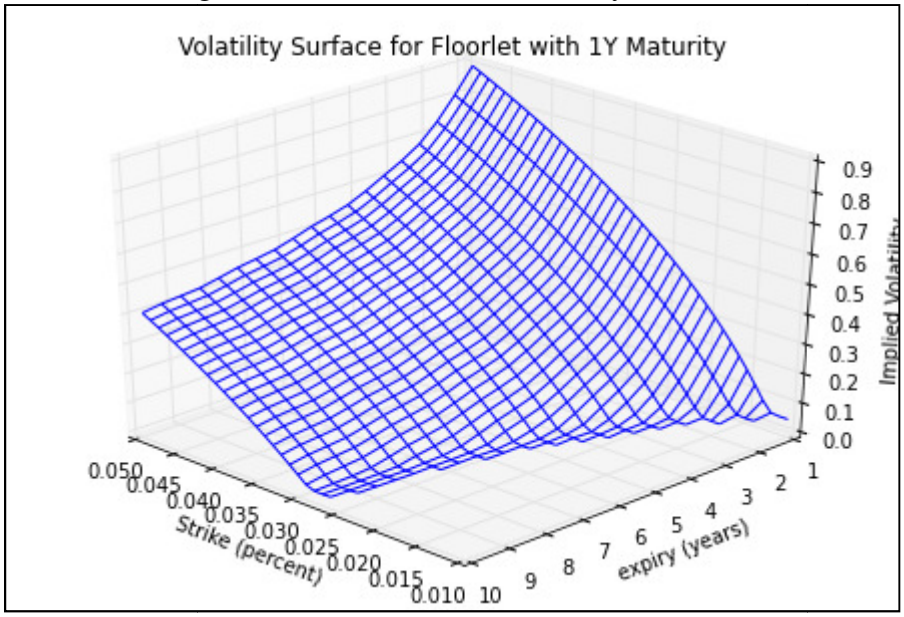


Figure 30: Sobol Floorlet Volatility Surface



The volatility surfaces of the Sobol implementation demonstrate much less variance in pricing as compared to the Numpy implementation as is evident by a lack of a

volatility estimate where caplet and floorlet prices were consistently zero over all simulations. The Numpy implementation, on the other hand, demonstrates some degree of a volatility smile – particularly for floorlets. This result can be attributed to the greater variance in paths for the short term forward rate over the course of the Numpy simulation.

Both Numpy and Sobol exhibit increasing implied volatility for caplets of increasing maturities across all strikes. For floorlets the opposite is true – implied volatility is generally lower for longer dated maturities. This behavior is consistent with the general trend of increasing short term rates over the simulation period for both Numpy and Sobol.

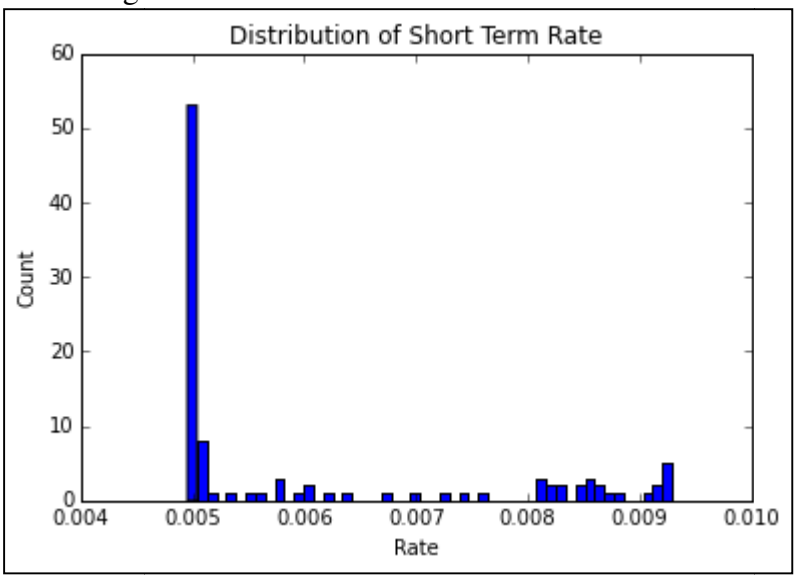
VASICEK MODEL

Zero-coupon bond prices from the HJM model were compared with those derived from the closed-form solution of a calibrated Vasicek model:

$$dr = \lambda(\mu - r)dt + \sigma dW_t$$

The same history of short term rates were used in the calibration – January 2012 to December 2013. The histogram below shows that the short term forward rate ranged from between 0.5% and 0.9% with the majority of the data points clustered around 0.5%.

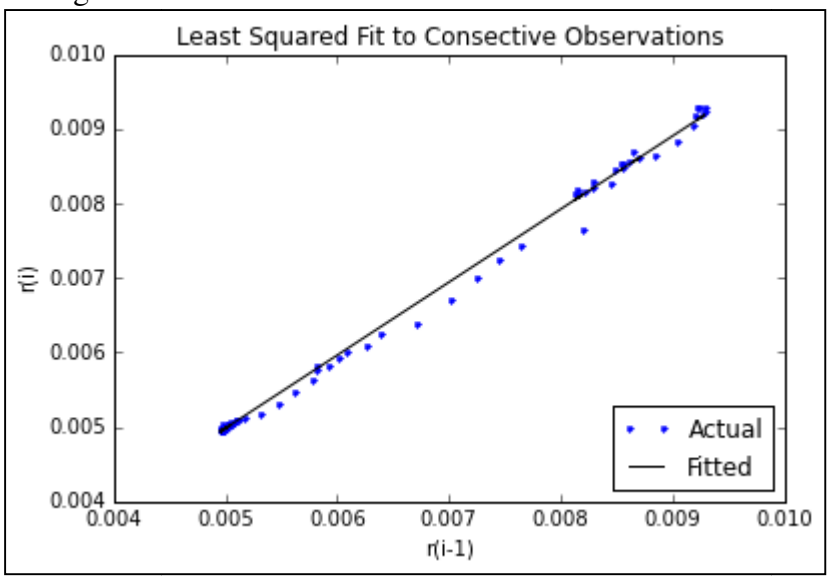
Figure 31: Histogram of Short Term Forward Rates – Jan 2012 to Dec 2013



The Vasicek model was calibrated using an ordinary least squares approach as outlined by Thijs van den Berg at Sitmo[7]. The figure below illustrates a line of best fit when regressing r_{i-1} to r_i. Line of best fit:

$$r_i = 0.00008 + 0.98001 r_{i-1}$$

Figure 32: Line of Best Fit for Short Term Forward Rates



[7] http://www.sitmo.com/article/calibrating-the-ornstein-uhlenbeck-model/

The calibration involves taking the slope, intercept and standard error from the line of best fit and calculating the parameters for the Vasicek model as follows:

$$\lambda = -\frac{\log a}{\delta}$$

$$\mu = \frac{b}{1-a}$$

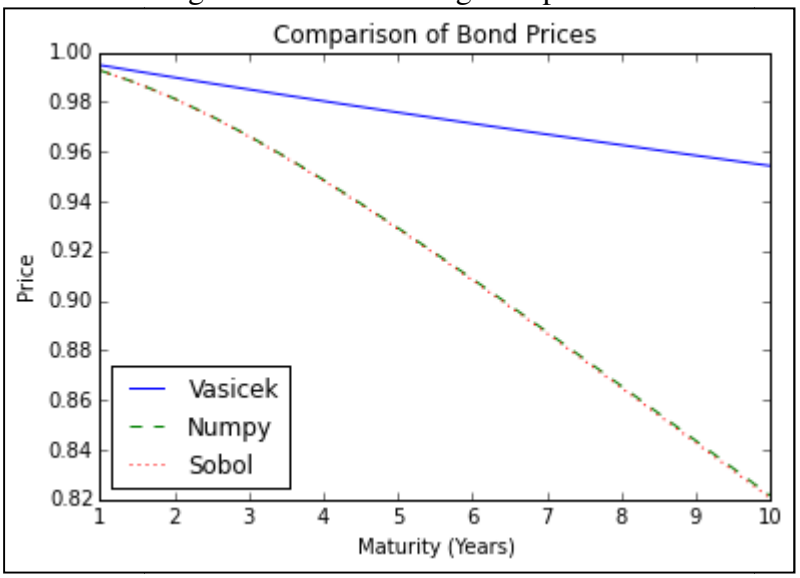
$$\sigma = sd(\epsilon)\sqrt{\frac{-2\log a}{\delta(1-a^2)}}$$

The resulting calibrated parameters are:

$$\lambda = 0.142, \qquad \mu = 0.004, \qquad \sigma = 0.0002$$

Bond prices were computed using the pricing equation for the Vasicek model along with the calibrated parameters. Below is a comparison of risk-free zero-coupon bonds over a range of maturities for Vasicek as well as the two HJM implementations: Numpy and Sobol.

Figure 33: Bond Pricing Comparison

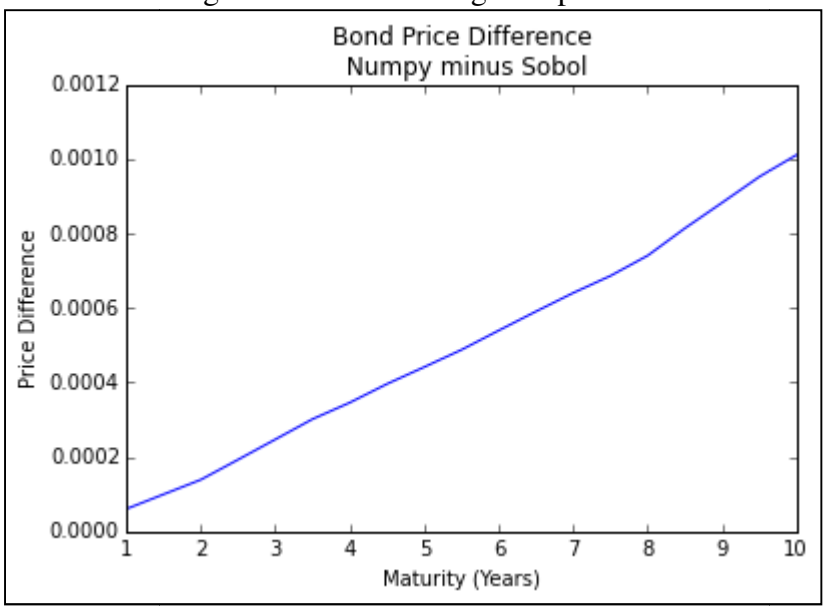


The figure clearly shows that bond pricing for the HJM implementation was more aggressive than that of the Vasicek model. This difference is expected to some degree since the HJM model incorporates movements over the entire forward rate curve with the result that there is a gradual but consistent increasing trend in the short term forward rate over the simulation time.

The Vasicek model does not incorporate this information. Moreover, the Vasicek model was calibrated using the same forward rate data as that of the HJM model. This data only reflects a low rate regime with little volatility in the short term rate. Had the Vasicek model been calibrated over a longer time period spanning both high and low rate regimes, the calibrated lambda, mu, and sigma parameters would have all likely been higher. The result would be bond prices more in line with those of the HJM implementation.

To gain a more clear comparison between the Numpy and Sobol zero-coupon bond prices, a plot of the differences was constructed.

Figure 34: Bond Pricing Comparison



The plot clearly shows that Numpy tends toward higher bond prices with increasing maturity. Consequently, it seems as though the Numpy implementation simulates slightly lower forward rates over the course of the simulation period. Further research we be required to see if this is truly an anomaly or a difference that can be decreased with an increasing number of simulations.