2017 Applied Finance Project
Convertible Bond Pricing Strategy
Final Report

Hang (Joy) Che
Nima Nader
Danny Ferdman
Tsung-Yen (Daniel) Ho

Company: Stout Risius Ross
Faculty Advisor: Prof. Francis Longstaff

December 15, 2017

Contents:

1. Project Description

1.1 Industry Background

Convertible bonds are corporate debt securities that provide the holder the right to forgo future coupon and/or principal payments and convert to a specified number of shares of common stock instead. The market of convertible bonds has been increasing. As of December 2016, the U.S. convertible market has $207.5 billion market capitalization, and convertible bonds make up 74%.

Because convertible bonds have equity-like and bond-like characteristics, convertibles bonds have became a popular security to invest. Convertible bonds help broaden investor base by providing flexible capital structure. In addition, a corporation issues convertible bonds to take advantage of reduced interest rates, since the presence of the conversion option provides upside potential for bondholders, and these bonds tend to demand lower interest rates compared to standard nominal bonds. Also, convertible bonds have tax reduction of interest compared with equity, which lowers the cost of capital for a company. Because of these benefits, many fast growing companies tend to invest in convertible bonds to raise capitals.

1.2 Project Goals

The objective of this project is to build convertible bond pricing models and compare the results of the models to market prices. Since convertible bonds have equity-like and bond-like characteristics, convertible bonds can only be accurately valued by considering both equity and fixed-income parts simultaneously. One of the major goals is to find an efficient pricing method that is consistent with observed market prices. The models we chose require complicated modeling, so we decided to code in R (R codes are provided in the Appendix). Because Stout prefers model presentations in Excel, we created Excel spreadsheets that contain all inputs and will call the R script for valuation, and our R code will record the results in the excel spreadsheet again.

To better understand different methodologies to price convertible bonds, we reviewed relevant academic and industry papers on this topic. In addition, we actively discussed with our faculty advisor and Stout. Our team also attended a seminar presented by EY Complex Securities Valuation team. For convertible bonds pricing, our team think that finding the correct volatility is one of the important steps we should take to get our model prices match with market prices. Therefore, we decided to conduct research on Goldman Sachs model and Stochastic Volatility model, which illustrate both the constant volatility and stochastic volatility scenarios. For the Goldman Sachs model besides just replicating the pricing models mentioned in the papers, we did extra research on predicting volatility, which will be discussed in detail in this paper. There is a lot of literature on convertible bonds. However, all the models we reviewed relied quite a bit on Monte Carlo simulation and less on arbitrage free pricing. Therefore, building a model for pricing and testing its effectiveness across multiple industries and terms is not a trivial matter.

2. Goldman Sachs Model

2.1 Literature Review

In the Goldman Sachs Quantitative Strategies Research Notes (1994), a foundational but widely used convertible bonds pricing model is discussed. The model can be used as a starting point when we build more complicated models in the future. Industry practitioners extend and modify this model to capture many different features of convertible bonds. In this Goldman Sachs's paper, it describes a binomial one-factor model to calculate convertible bonds' theoretical values.

The important starting point of the Goldman Sachs model is to derive the current theoretical value of the convertible bond from the current values of its underlying straight debt and equity. There are two ways to restructure the convertible bonds:

(1) a straight bond and a call option that you can exchange bond for equity;
(2) an equity plus a put option that you can exchange the equity for a straight bond, and a swap to maturity that you can exchange the bond's coupons for the equity's dividends.

The Goldman Sachs paper assumes fixed interest rates and volatility of the underlying stock, so the only varying parameter is the future price of the underlying stock. This is because the author experimented to prove that the price uncertainty is the main determination of option values in convertible bonds. In addition, the paper also assumes the distribution of the future stock prices is lognormal, and the credit spread for the issuer's straight bonds provides all the information needed on default risk.

Now we are going to discuss the detailed steps we take for the Goldman Sachs model. First, we build a stock tree that extends from the valuation date to the maturity of the convertible bond, as shown in Figure 1.

Second, on the maturity date compute the value of the convertible bond as the maximum of its fixed-income redemption value and its conversion value. The redemption value is sum of the par value and the coupon payment at each time period. For instance, at the 5^{th} time period, the redemption value is \$110, which is the sum of par \$100 and coupon \$10. In addition, at nodes where it pays to convert, define the probability of conversion to be 1, and 0 otherwise.

Third, move backwards in time down the tree, one level at a time. At each node within each level, define the conversion probability as the average of the probabilities at the two connected future nodes. Also find the corresponding risk-adjusted discount rate for each node within each level. If we convert the bond to stock at the node, the discount rate is the riskless rate 5%. If we do not convert the bond to stock, the discount rate is 10%, which is the sum of riskless rate 5% and risk spread 5%. Next, get the holding value defined as the discounted weighted average of the next period without conversion, put, or call in that time period.

Finally, compare the holding value to the payoff of the call. The company will call the bond if the payoff of the call is greater than the holding value, then we get a new holding value. Next, compare the new holding value with the payoff of the put. The investor will put the bond if the

payoff of put is greater than the holding value. Then, compare the holding value to the exercise value; the investor will convert if the exercise value is greater (Exercise payoff is the stock price. If you exercise, you forgo the coupon payment).

The discounted rate for every node is as follows: if a put exercised, the discount rate is risk-free rate. If a call exercised, the discount rate is sum of the risk-free rate and default risk spread. If the optimal decision is to hold the bond, the discount rate is the weighted average of the subsequent period discount rates. If the optimal decision is to exercise the conversion, the discount rate is risk-free rate.

After we built the model by following the procedures above, we first tested the hypothetical convertible bond provided by the paper (Table 1). Figure 1 below includes three sections: the call, put, and coupon payment for each time period, a stock tree with the required structure, and a corresponding convertible tree. As illustrated below, our model accurately produces the price of the convertible bond provided in the paper.

TABLE 1: A Hypothetical Convertible Bond

Principal:	$100
Coupon	10% per year [a]
Maturity:	5 years
Conversion ratio:	1
Calls:	$115 in year 2, declining by $5 every year to maturity
Puts:	$120 in year 3
Current stock price:	$100
Stock dividend rate:	0%
Volatility:	10% per year
Riskless rate:	5% per year
Stock loan rate:	5% per year
Credit spread:	500 basis points

a. Bond coupon, stock dividend yield and all interest rates are annually compounded.

Figure 1. Binomial Trees for Valuing the Convertible Bond

Time (years)	0	1	2	3	4	5
Calls	NA	NA	115	110	105	100
Puts	NA	NA	NA	120	NA	NA
Coupon Payments	10	10	10	10	10	10

Stock Tree

Each node shows stock price.

					205.24
				177.75	
			153.94		168.03
		133.32		145.53	
	115.47		126.04		137.57
100.00		109.15		119.15	
	94.53		103.19		112.64
		89.37		97.55	
			84.48		92.22
				79.87	
					75.50

Convertible Tree

Each node shows: convertible value, action taken, and conversion probability.

X: convert
P: put
C: call
H: hold
R: redeem

					205.24 X 1.0
				177.75 X 1.0	
			153.94 X 1.0		168.03 X 1.0
		133.32 X 1.0		145.53 X 1.0	
	130.30 H 0.5		130.00 P 0.0		137.57 X 1.0
126.82 H 0.25		125.00 C 0.0		119.15 X 1.0	
	123.64 H 0.0		130.00 P 0.0		112.64 X 1.0
		125.00 C 0.0		113.64 H 0.5	
			130.00 P 0.0		110.00 R 0.0
				110.00 H 0.0	
					110.00 R 0.0

2.2 Data Description

The data used for the Goldman Sachs model comes from two sources. First, we got convertible bond data and their underlying equity data from Bloomberg. Convertible bond data includes bond price, conversion ratio, maturity, credit spread to maturity, and coupon rate. The bond's underlying equity data includes stock price, 90-day implied volatility, current market capitalization, P/E ratio, and price to book ratio. When we were drafting this final report after our presentation, we added industry effect in the Goldman Sachs model, so data on the industry type of the underlying equity is also collected from Bloomberg. Second, risk-free rates are needed for the discounting purpose in the Goldman Sachs model. We used the daily treasury yield curve rates from the U.S. Department of the Treasury website. Instead of using linear interpolation, we calculated risk-free rates by conducting logarithm interpolations, and we also included this calculation process in our coded model.

All the data was downloaded to an excel spreadsheet through the Bloomberg add-in. No additional manipulation is done in Excel. The data is structured, cleaned and reshaped in R. This way the code is highly robust to additional data and unbalanced data inputs.

In order to have enough data to test our Goldman Sachs model, we included data from 2008 to 2017. After we deleted those data points with invalid price or credit spread, or those with insufficient bond volume under 100 per week, we now have over 60,000 convertible bond data points in total.

2.3 Model Validation

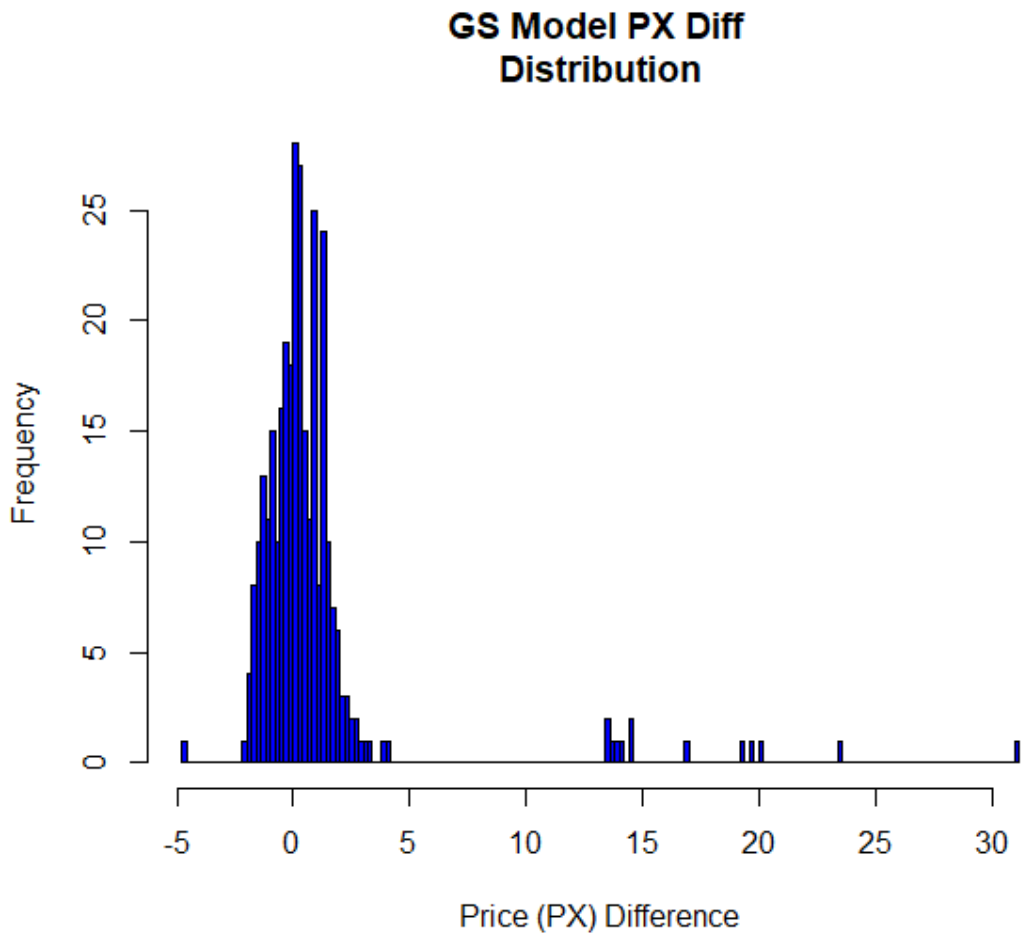
The Goldman Sachs model uses constant volatility assumption, and using the correct volatility is essential in the pricing process. Before we get into the details on the volatility prediction, we first want to make sure our model is valid. In the model validation process, we want to test the most liquid call options because we can directly use the available 90-day implied volatility provided by Bloomberg. We restricted our model validation data to those convertible bond data points with time to maturity under 1 year, and we used the 90-day implied volatility from at the money call options for the market expected time to maturity equity volatility. For model validation we ended up with 315 convertible bond data points in total.

After we restricted our dataset into a data sample that we could use in the model validation. We calculated convertible bond prices using the Goldman Sachs model, and compared the model prices with the bonds' market prices by plotting the distribution as shown in Figure 2. Excluding the distressed convertible bond (in Appendix), 95% of convertible bond data points have price difference between the Goldman Sachs model and the market price within 2%.

Those distressed convertible bond data points which go beyond 10% all come from one convertible bond "ICON 1 ^{1}2 03/15/18". The incorrect pricing can be explained fact that ICON's equity price dropped from 6.00 to 1.51 in November 2017 (in Appendix). The data sources for this particular bond were not updated in Bloomberg, specifically the credit spread, which led a

bond overvaluation. Since all the testing was done systematically we did not bother fixing the data error as it was properly flagged and identified.

Figure 1. Goldman Sachs Model Price Difference Distribution for Model Validation



2.4 Volatility Prediction

One of the main inputs, if not the most important input, into the Goldman Sachs model is the volatility value for the length of the convertible bond. Getting implied volatility from traded options is an acceptable method even when considering all the caveats. However, options implied volatility is only possible when the options maturity is close enough to the bond maturity term. Here lies the crux of the problem: sufficiently liquid exchange traded options have maturities of at most one year. Our solution for forecasting long-term volatility is creating a linear regression model. From the EY presentation regarding convertible bonds we got the idea of using an implied volatility haircut. The concept is to apply a certain haircut to match market expectations. We tested the assumed assumption on that long-term volatility and by extension volatility haircut, is highly related to expected returns. From Fama-French 3 factor model we

added log market capital, price to book. We also added price to earnings ratio, 90-day historical volatility, and moneyness (defined as stock price minus conversion price divided by stock price). Our volatility haircut regression provides outstanding results: the R squared value and standard error are 0.962 and 0.0689 respectively. The full regression output is shown in Figure 2. A big difference is achieved by including industry fixed effects as can be evidence by the regression results when the industry dummy is omitted with the R squared value and standard error are 0.649 and 0.190 respectively.

This volatility model is incredibly useful for making the Goldman Sachs model applicable. For publicly listed companies getting implied volatility is rather straight forward and it is also assumed that getting implied volatility for private firms is not too complicated.

Figure 2. Volatility Haircut Regression

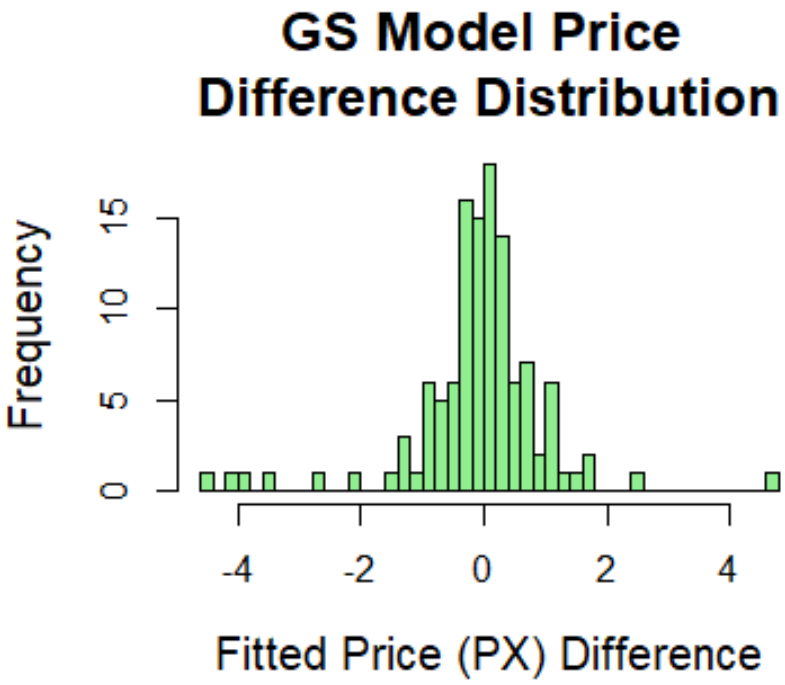
```
Call:
lm(formula = (solvedVol/(`3MO_CALL_IMP_VOL`)) ~ log(CUR_MKT_CAP) +
    {
        sign(moneyness) * (abs(moneyness)/StockPX)
    } + TtoMaturity + (PE_RATIO) + (PX_TO_BOOK_RATIO) + (VOLATILITY_90D) +
    BICS)

Residuals:
      Min        1Q    Median        3Q       Max
-0.194521 -0.027502  0.000966  0.037087  0.188302

Coefficients:
                                                  Estimate Std. Error t value Pr(>|t|)
(Intercept)                                     -0.7369506  0.1374373  -5.362 5.97e-07 ***
log(CUR_MKT_CAP)                                 0.2656515  0.0145523  18.255  < 2e-16 ***
{\n    sign(moneyness) * (abs(moneyness)/StockPX)\n} -0.4409920  0.0543089  -8.120 1.90e-12 ***
TtoMaturity                                     -0.0561925  0.0069365  -8.101 2.08e-12 ***
PE_RATIO                                        -0.0009592  0.0001356  -7.071 2.81e-10 ***
PX_TO_BOOK_RATIO                                -0.0341930  0.0046277  -7.389 6.28e-11 ***
VOLATILITY_90D                                  -0.9871788  0.1955172  -5.049 2.20e-06 ***
BICSBldg-Residential/Commer                      0.2523697  0.0709172   3.559 0.000589 ***
BICSBuilding-Heavy Construct                     0.2313367  0.0515013   4.492 2.03e-05 ***
BICSCommercial Services                          0.1991473  0.0412306   4.830 5.35e-06 ***
BICSComputers-Other                              0.3852111  0.0776061   4.964 3.12e-06 ***
BICSComputers-Peripher Equip                     0.5506840  0.1942812   2.834 0.005630 **
BICSE-Commerce/Products                          0.1226126  0.0539465   2.273 0.025338 *
BICSE-Commerce/Services                         -0.1644240  0.0534400  -3.077 0.002749 **
BICSElectronic Compo-Misc                       -0.1988010  0.0595606  -3.338 0.001216 **
BICSElectronic Compo-Semicon                    -0.0519052  0.0494761  -1.049 0.296852
BICSFootwear&Related Apparel                    -0.9774166  0.1416144  -6.902 6.18e-10 ***
BICSGold Mining                                  0.2592583  0.0509612   5.087 1.88e-06 ***
BICSLasers-Syst/Components                       0.2021149  0.0590507   3.423 0.000923 ***
BICSMedical-Biomedical/Gene                      0.0710693  0.0520290   1.366 0.175248
BICSMedical-Drugs                               -2.2611493  0.2789876  -8.105 2.05e-12 ***
BICSMetal Processors&Fabrica                    -0.3762363  0.0606407  -6.204 1.51e-08 ***
BICSRetail-Pawn Shops                           -0.1906219  0.0681332  -2.798 0.006254 **
BICSSemiconductor Equipment                     -0.1899542  0.0612733  -3.100 0.002560 **
BICSTransport-Equip&Leasng                      -0.1018989  0.0581854  -1.751 0.083195 .
BICSWeb Hosting/Design                           0.0202023  0.0615117   0.328 0.743325
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.06868 on 93 degrees of freedom
Multiple R-squared:  0.9621,  Adjusted R-squared:  0.9519
F-statistic: 94.39 on 25 and 93 DF,  p-value: < 2.2e-16
```

2.5 Analysis and Results

We calculated convertible bond prices in our Goldman Sachs model with forecasted volatility, and compared the model prices with the bonds' market prices by plotting the distribution below

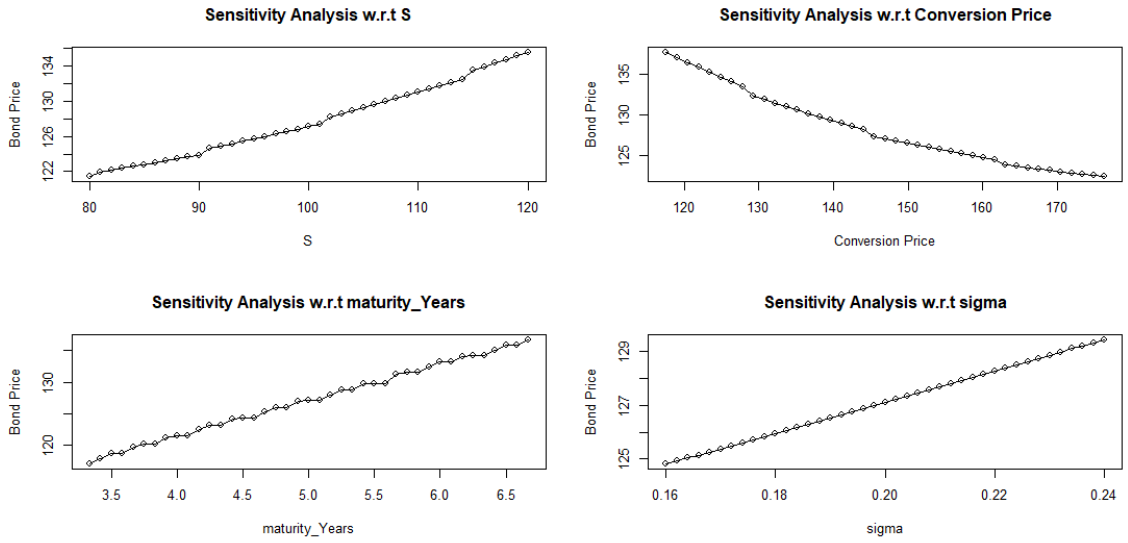
in Figure 3. 92% of convertible bond date points have fitted price difference between the Goldman Sachs model and the market price within 2%. Thus, we established that using linear regression for forecasted volatility only marginally widens the error term.

Figure 3. Goldman Sachs Model Price Difference Distribution



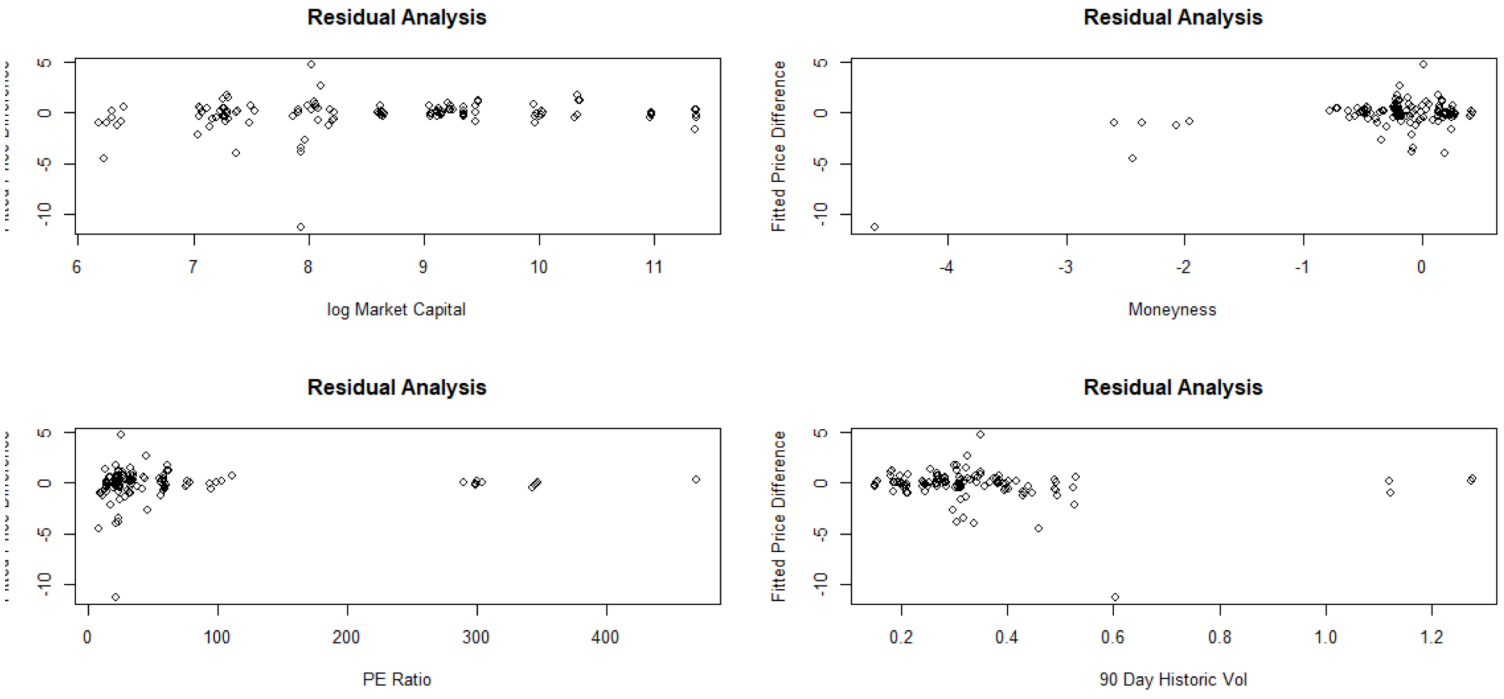
In order to test how the changes of our input variables affect on the convertible bond price, we also conducted sensitivity analysis to see how convertible price changes with changes in different input variables, as shown in Figure 4. It is clear that the convertible bond price increases when the underlying stock price, maturity, or volatility increases. The convertible bond price decreases when the conversion price increases.

Figure 4. Sensitivity Analysis



Furthermore, we inspected all the residuals from the model to see if there were non-linearities or heteroskedastic error terms. Four of the graphs are shown in Figure 5 for illustration purpose. From a simple visual inspection, we see that there are no heteroskedasticity and all the error terms look like white noise. Please note that there are outliers in the data which are separated from the main data clusters but they are not biased.

Figure 5. Residuals Analysis



2.6 Summary

From the analysis and results, we conclude that our Goldman Sachs model matches market prices with good precisions. Furthermore, to utilize the pricing model for new bonds or bonds without attainable term expected volatility we developed a model to extract volatility based on commonly found factors. Stout can use this model to price or compare their clients' convertible bond contracts both for publicly traded companies and privately held companies. In addition to pricing, the Goldman Sachs model has additional uses. We can predict volatilities for equities over long horizons by applying the volatility haircut as implied by the linear regression model. This is highly useful including for pricing of various types of options. Moreover, the model can be used to test mispricing relative to the market. The uses of this comparison can be extended for anything between credit spreads to statistical arbitrage. We saw two cases where the market prices didn't match the model price. We didn't have enough time to identify the mispricing sources, but such analysis is highly valuable albeit outside the scope of this project.

There are further improvements that can be done with our current Goldman Sachs model to improve pricing accuracy in the future. First, we can increase our sample size for the volatility haircut regression analysis. This is tricky and requires careful data validation but can be done. Second, since some of the client companies at Stout are privately held the use of comparables needs to be tested and potentially adjusted.

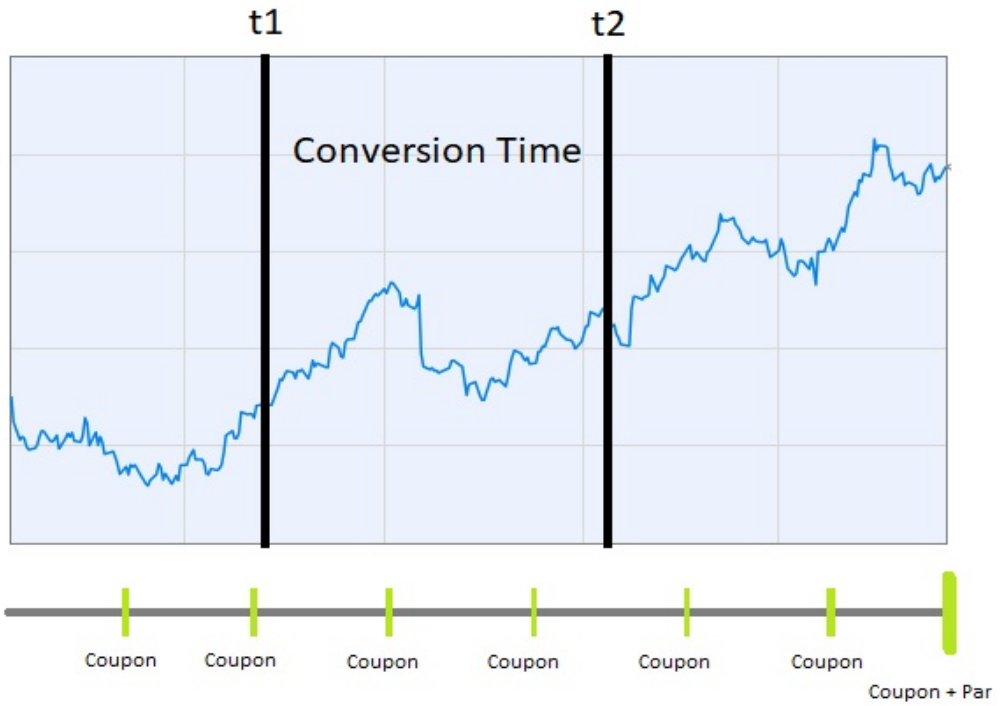
3. Stochastic Volatility Model

3.1 Model Description

The scope of this section is to design a model that utilizes the MFE knowledge to build a new way of pricing convertible bonds. When a stock price moves downward an investor has no incentive to convert into equity shares and therefore the investor would only collect coupons along time and par at maturity, but when a stock price moves upward to a point that the investor would benefit from conversion, the investor would convert if the future cash flow is less than what the investor can earn at time of conversion. Based on the logic explained, the price of the convertible bond should always be more than a bond price alone and is highly dependent to stock path. One of the simplest and best way of pricing a path dependent option is to use Monte-Carlo simulations. A model must be used to simulate a stock path and therefore choosing a model would be an important aspect to this section.

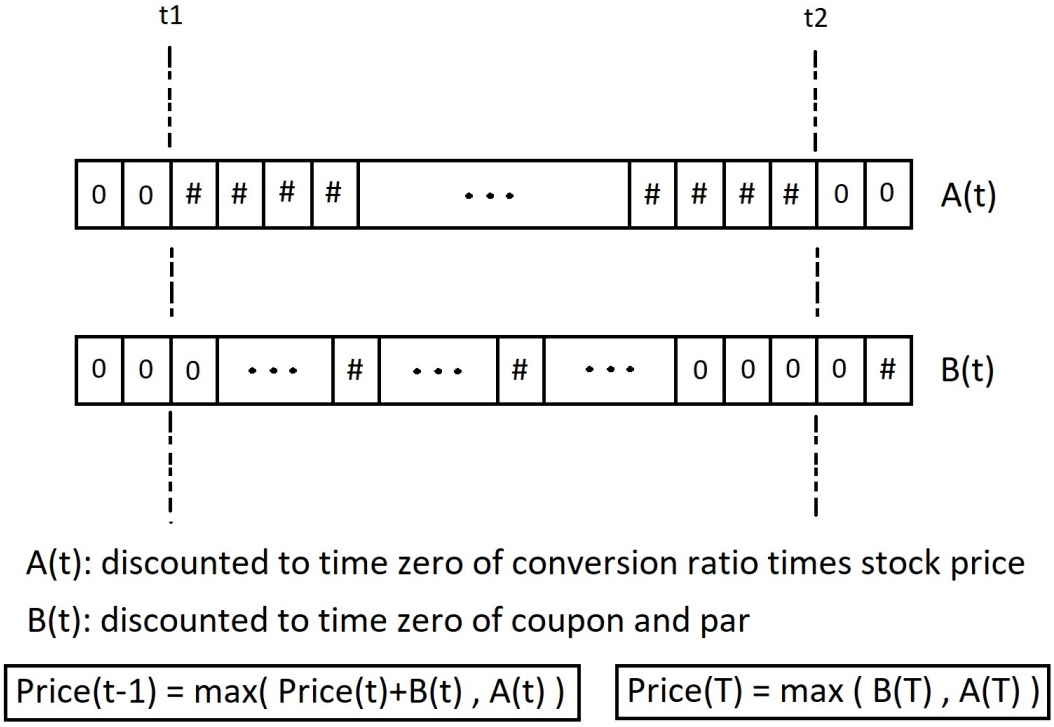
We assume that a stock path with coupon payments as shown in Figure 5 below has realized in the market and the goal is to find a true value of this convertible bond. Investors have the option to convert at any day in between time t1 and t2 where conversion can only happen.

Figure 5. Realized Stock Path with Coupon Payments



To appropriately price such a convertible bond we can generate two vectors as shown in Figure 6 below. The first vector contains daily stock prices multiplied by conversion ratio discounted to time zero called A(t). Note that we only allowed to convert at time in between t1 and t2, therefore the vector of A(t) should have zeros in other times. The second vector is a vector of all zeros with only discounted coupons and par to time zero called B(t). Backward pricing technique needs to be done to price this convertible bond. First, we take the price as a maximum of stock converted and discounted at maturity time, A(T) with discounted coupon and par at maturity time, B(T) and we can call this the discounted price at maturity, Price(T). Second, we compare A(t) by Price(t)+B(t) and we take the price at time t-1 to be the maximum of the two and we do this process all the way to time zero.

Figure 6. Methodology to Price Convertible Bonds



The above technique assures to collect coupons prior to conversion and incorporates it to price the convertible bond. The stock path A(t) can be changed due to different contracts. If there exist call-ability or put-ability, the logic of pricing stays the same and only the values in vector A(t) change accordingly.

In the Goldman Sachs model, two main assumptions we considered are the constant volatility and interest rate. However, in this pricing technique we use models that do not assume constant volatility and interest rate. The interest rate model used is CIR model because of its inherent simplicity and it produces positive interest rates, which were historically the case in the United States. The model used to generate stock paths is the Heston model, it is a standard stochastic volatility price dynamics workhorse. These models are described in the following sections.

3.1.1 Cox-Ingersoll-Ross Model (CIR)

The CIR model is used to describe the evolution of short rates defined as the following stochastic differential equation:

$$dr(t) = k\big(\theta - r(t)\big)dt + \sigma\sqrt{r(t)}dW$$

Where k is the mean reversion speed, θ is the long term mean, σ is the volatility, and r_0 is the starting value for short rate CIR process. We will be calibrating CIR process and pricing contracts using this model.

Notice that the diffusion term is proportional to $\sqrt{r(t)}$ ensures that the process has positive value if the constrain $2\,k\,\theta > \sigma^2$ (Fellor condition) is satisfied. Also, the σ is multiplied by the square of the short rate, its value is not comparable with the volatility parameter of the Vasicek model.
Overall, the price in t of a zero-coupon bond with maturity T is giving by

$$P(t,T) = A(t,T)e^{-B(t,T)r(t)}$$

Where:

$$A(t,T) = \left[\frac{2he^{\frac{(k+h)(T-t)}{2}}}{2h + (k+h)(e^{(T-t)h} - 1}\right]^{\frac{2k\theta}{\sigma^2}}$$

$$B(t,T) = \frac{2(e^{(T-t)h} - 1)}{2h + (k+h)(e^{(T-t)h} - 1)}$$

With $h = \sqrt{k^2 + 2\sigma^2}$.

To optimize such model, we use daily treasury yield curve rates. The process to optimize are as follow:
1. We assign constant values for our three variables k, θ and σ.
2. From historical data (CMT) and considering P(0,T) we can find S_0 for each day for example we can use 1mo yield curve value to calculate S_0.
3. Using calculated S_0 we can calculate appropriate yield at time T using P(0,T). For example, we can use S_0 to calculate yields at time 3mo, 6mo, 1year etc.
4. A cost function must be defined as the sum of all squared of differences between calculated yields and actual CMT data.
5. We optimize the model by changing our constant values for k, θ and σ to minimize the cost function.

3.1.2 Heston Model

Heston model is a stochastic process model that its underlying asset follows the following risk neutral dynamics.

$$dS_t = rS_t dt + \sqrt{V_t}S_t dW_t^1$$
$$dV_t = a(V - V_t)dt + \eta\sqrt{V_t}dW_t^2$$
$$dW_t^1 dW_t^2 = \rho dt$$

Where:
S_t is the price of the underlying asset at time t
R is the risk-free rate
V is the long-term variance
V_t is the variance at time t
a is the variance mean reversion speed

η is the volatility of variance process
dW_t^1 and dW_t^2 are two correlated Weiner process
ρ is correlation coefficient

Among all types of different volatility models, Heston model actually provides us with the desirable properties of mean-reversion process for volatility and correlated shocks between asset returns and volatility. However, the most important thing is that Heston model provides a versatile modelling framework for many specific characteristics that are observed in the behavior of financial market. On top of that, η controls the kurtosis and ρ sets its asymmetry.

There are two ways to optimize such model to find our five constants variables V, a, η, ρ and V_0. One way is to use European option prices on the specific company, and another way is to use historical data to optimize our constants.

To optimize using European option prices there exist a closed form solution for Heston model. The reason is that the probability value can be obtained by using the dynamic formulas from the above. However, the probabilities are hard to be calculated due to the transition densities. The characteristic function for Heston model are shown as following:

$$\psi_{\ln(S_t)}^{Heston}(w) = e^{[C(t,w)V + D(t,w)V_0 + iw\ln(S_0 e^{rt})]}$$

$$C(t,w) = a[r_- \bullet t - \frac{2}{\eta^2}\ln\left(\frac{1-ge^{-ht}}{1-g}\right)]$$

$$D(t,w) = r_- \frac{1-e^{-ht}}{1-ge^{-ht}}$$

$$r_\pm = \frac{\beta \pm h}{\eta^2};\ h = \sqrt{\beta^2 - 4\alpha r} \quad (3.3)$$

$$g = \frac{r_-}{r_+}$$

$$\alpha = -\frac{w^2}{2} - \frac{iw}{2};\ \beta = a - \rho\eta iw\ ;\ r = \frac{\eta^2}{2}$$

By knowing the characteristic function, we can simply price a European call options using the following pricing technique.

$$C_0 = S_0\, \pi_1 - e^{-rT} K\, \pi_2$$

Where:

$$\pi_1 = \frac{1}{2} + \frac{1}{\pi}\int_0^{inf} Re[\frac{e^{-iw\ln(k)}\Psi_{\ln S_t}(w-i)}{iw\Psi_{\ln S_t}(-i)}]\, dw$$

$$\pi_2 = \frac{1}{2} + \frac{1}{\pi}\int_0^{inf} Re[\frac{e^{-iw\ln(k)}\Psi_{\ln S_t}(w-i)}{iw}]\, dw$$

To optimize Heston model using European call option prices we use the following steps:
1. We assume constant values for our 5 variables V, a, η, ρ and V_0.

2. We calculate European call option prices using market data information.
4. A cost function must be defined as the sum of all squared of differences between calculated European call prices and actual European call prices from data.
5. We optimize the model by changing our constant values for V, a, η, ρ and V_0 to minimize the cost function.

To optimize Heston model using historical data we use the following steps:
1. We assume constant values for our 5 variables V, a, η, ρ and V_0.
2. for each day we can calculate the 3 months and 6 months stock prices.
4. A cost function must be defined as the sum of all squared of differences between calculated stock price and actual stock price from data.
5. We optimize the model by changing our constant values for V, a, η, ρ and V_0 to minimize the cost function.
6. We repeat the above process 10 times (because each time we use random number generator for our calculations) and we average the values calculated for V, a, η, ρ and V_0 to find our constant values.

3.2 Data Description

The data used to optimize CIR model are daily treasury yield curve rates from U.S. Department of the Treasury website. We downloaded call option data and 1000 convertible bond prices for Tesla company from Bloomberg. Call options data includes spot price, maturity, strike, interest rate calculated from optimized CIR model, bid, ask and mid-price by averaging the bid and ask prices. Convertible bond data includes bond price, conversion ratio, maturity, current market price, credit spread to maturity, and coupon rate. we calculated risk-free rates using CIR model and incorporated that accordingly into our code. We used 5 years historical daily stock price for Tesla from Nasdaq website.

3.3 Analysis and Results

First CIR model has been optimized to calculate our three constant values k, θ and σ. The optimization process is explained in section 3.1.1 and a sample of data used can be found in appendix. Also, the code is attached in the code section. The result of this optimization for our constants are shown in Table 2 below:

Table 2: Optimized CIR Variables

k	θ	σ
0.0929	0.1274	0.0686

Using above values, we calculated the yields from CIR model on 11/20/2017 and compared the results with actual treasury yield curve and a portion of these results is shown in Table 3 below. Please see appendix for full result.

Table 3: CIR Estimated Results vs. Treasury Yield

	3month	1year	2year	7year
Treasury Yield	0.013	0.0162	0.0177	0.0226
CIR Estimate	0.011	0.0133	0.0156	0.0245
% Error	12.75	18.16	11.76	8.52

This result indicated that CIR model is not a good fit for CMT data and therefore it is better to use other models such as two-factor models like G2++ model.

Using CIR model, we generate appropriate interest rate paths and incorporated that to optimize five variables V, a, η, ρ and V_0 in the Heston model for Tesla. The optimization method is explained in section 3.1.2 and a sample of data used can be found in appendix. The results calculated for our constants are shown in table 4 and 5:

Table 4: Optimized Heston Variables Using Tesla European Call Prices

V_0	ρ	η	V	a
0.1101	-0.4564	1.1414	0.1888	6.8825

Table 5: Optimized Heston Variables Using Tesla Historical Stock Prices

V_0	ρ	η	V	a
0.196	-0.567	0.97	0.123	0.981

The comparison of the above optimized results indicate that optimization using different sources of data can have diametrically different results. And this can be an issue to appropriately pricing our convertible bonds.

Since Heston model is a model under risk neutral probability there is no need to use call options data to optimize our 5 variables, historical data should be used to price convertible bonds. By using our optimized data in Table 2 and Table 5 and 1000 convertible bonds for Tesla, we calculated the percent difference between model prices and market prices for validation. Two methods been used to price convertible bonds, one without default risk as indicated in Figure 7 below and one with default risk as indicated in Figure 8 below.

Figure 7: Percent Price Difference Without Default Risk

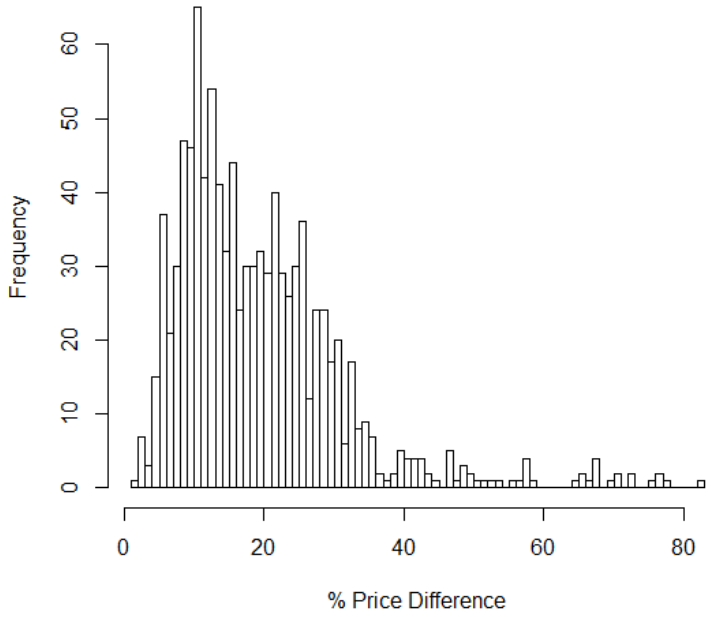
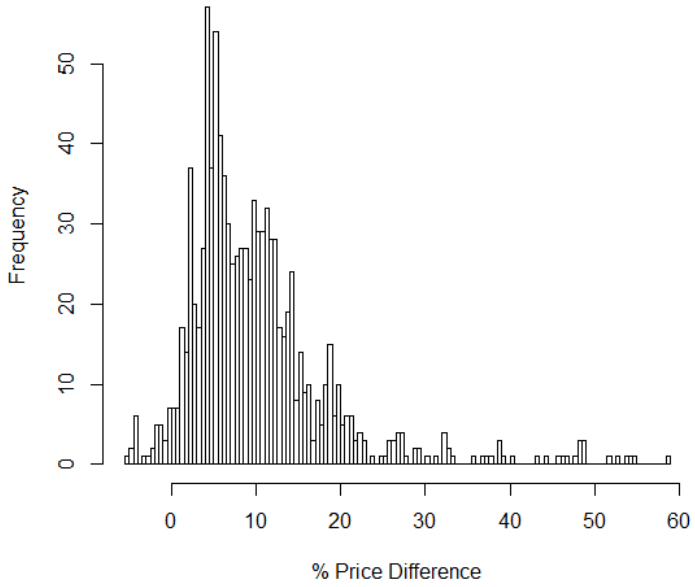


Figure 8: Percent Price Difference With Default Risk



Many papers just added default risk to risk-free rate and calculated discount factor which we use the same technique when we are including default risks, but as a fact we know as the stock prices goes up the default risk goes down, therefore there must be a correlation between default risk and

stock prices. However, adding constant default risk to risk-free rate to discount can give us some information regarding the importance of a default risk.
Figure 7 shows that all calculated convertible bonds prices are higher than actual market prices and this is exactly what we expected since we are not including the default risks, and as far as model validation 60.1% of data are within 20% of accuracy to market prices. Figure 8 shows that 91.5% of our data are within 20% of accuracy to market prices. We can see that including default risk has improved our valuations.

3.4 Summary

Based on the results above we can see that using multi variable plays an important rule in pricing convertible bonds. It is especially sensitive over which time horizons the optimization is calculated over. We also noticed how important default risk is to price convertible bonds, and incorporating that appropriately can be significant to find an accurate price. The pricing technique using Monte-Carlo simulation is a valid technique and can be used by any models. This model should also do well in pricing if we use accurate optimized constants for Heston model and using two-factor model like G2++ for interest rates.

References

Goldman Sachs Quantitative Strategies Research Notes, 1994, Valuing Convertible Bonds as Derivatives.
Kostas, T., Chris, F., 1998. Valuing Convertible Bonds with Credit Risk. Journal of Fixed Income.
Goldman Sachs Quantitative Strategies Research Notes, 1994, Enhanced Numerical Methods for Options with Barriers.
Ricardo C., 2014. An Analysis of the Heston Stochastic Volatility Model: Implementation and Calibration using Matlab.
Fairmat. CIR Model Version 1.0.

Appendix

ICON 1 [1]2 03/15/18:

Sample data used to optimize CIR model:

Select type of Interest Rate Data

Daily Treasury Yield Curve Rates ▼ Go

Select Time Period

Current Month ▼ Go

Date	1 Mo	3 Mo	6 Mo	1 Yr	2 Yr	3 Yr	5 Yr	7 Yr	10 Yr	20 Yr	30 Yr
11/01/17	1.06	1.18	1.30	1.46	1.61	1.74	2.01	2.22	2.37	2.63	2.85
11/02/17	1.02	1.17	1.29	1.46	1.61	1.73	2.00	2.21	2.35	2.61	2.83
11/03/17	1.02	1.18	1.31	1.49	1.63	1.74	1.99	2.19	2.34	2.59	2.82
11/06/17	1.03	1.19	1.30	1.50	1.61	1.73	1.99	2.17	2.32	2.58	2.80
11/07/17	1.05	1.22	1.33	1.49	1.63	1.75	1.99	2.17	2.32	2.56	2.77
11/08/17	1.05	1.23	1.35	1.53	1.65	1.77	2.01	2.19	2.32	2.57	2.79
11/09/17	1.07	1.24	1.36	1.53	1.63	1.75	2.01	2.20	2.33	2.59	2.81
11/10/17	1.06	1.23	1.37	1.54	1.67	1.79	2.06	2.27	2.40	2.67	2.88
11/13/17	1.07	1.24	1.37	1.55	1.70	1.82	2.08	2.27	2.40	2.67	2.87
11/14/17	1.06	1.26	1.40	1.55	1.68	1.81	2.06	2.26	2.38	2.64	2.84
11/15/17	1.08	1.25	1.39	1.55	1.68	1.79	2.04	2.21	2.33	2.58	2.77
11/16/17	1.08	1.27	1.42	1.59	1.72	1.83	2.07	2.25	2.37	2.62	2.81
11/17/17	1.08	1.29	1.42	1.60	1.73	1.83	2.06	2.23	2.35	2.59	2.78
11/20/17	1.09	1.30	1.46	1.62	1.77	1.86	2.09	2.26	2.37	2.60	2.78
11/21/17	1.15	1.30	1.45	1.62	1.77	1.88	2.11	2.27	2.36	2.58	2.76
11/22/17	1.16	1.29	1.45	1.61	1.74	1.84	2.05	2.22	2.32	2.57	2.75

Wednesday Nov 22, 2017

Optimized variable calculated for CIR model, the actual result from the code:

Filter

kk	sigma	rbar
0.09291284	0.127362	0.0686392

Comparison between CIR model results and treasury yield curve on 11/20/2017

Filter

	Date	1 Mo	3 Mo	6 Mo	1 Yr	2 Yr	3 Yr	5 Yr	7 Yr
actual	11/20/17	0.0109	0.013	0.0146	0.0162	0.0177	0.0186	0.0209	0.0226
CIR Estimate	NA	0.0109	0.0113425597042397	0.0119947446354736	0.0132574067316529	0.01561879298689	0.0177693615356185	0.0214876001306962	0.0245247397747119
% error	NA	0	12.7495407366177	17.8442148255233	18.1641559774512	11.7582317124859	4.46579819559946	2.81148387892919	8.51654767571637

Sample of data used to optimize Heston Model:

Spot	Maturity	Strike	Interest rate	Mid	Bid	Ask
308.77	2.150793651	300	0.01595632	81.4	79.8	83
308.77	2.150793651	305	0.01595632	79.25	77	81.5
308.77	2.150793651	310	0.01595632	77.1	76	78.2
308.77	2.150793651	315	0.01595632	75.25	73	77.5
308.77	2.150793651	320	0.01595632	73.25	71	75.5
308.77	0.242063492	300	0.01132163	29.55	28.5	30.6
308.77	0.242063492	305	0.01132163	26.675	26.2	27.15
308.77	0.242063492	310	0.01132163	24	23.65	24.35
308.77	0.242063492	315	0.01132163	21.65	21.2	22.1
308.77	0.242063492	320	0.01132163	19.525	19.15	19.9
308.77	1.158730159	290	0.01364671	65.4	63.6	67.2
308.77	1.158730159	310	0.01364671	56.15	54.7	57.6
308.77	1.158730159	330	0.01364671	47.625	46.75	48.5
308.77	0.571428571	300	0.01217852	42.975	42.05	43.9
308.77	0.571428571	305	0.01217852	40.8	40.25	41.35
308.77	0.571428571	310	0.01217852	38.4	37.9	38.9
308.77	0.571428571	315	0.01217852	36.125	35.55	36.7
308.77	0.571428571	320	0.01217852	33.85	33.35	34.35
308.77	0.099206349	305	0.01094242	13.75	13.5	14
308.77	0.099206349	307	0.01094242	12.4	12.3	12.5
308.77	0.099206349	310	0.01094242	11.15	11	11.3
308.77	0.099206349	312	0.01094242	9.9	9.7	10.1
308.77	0.099206349	315	0.01094242	8.9	8.8	9

Optimized constants for Heston model using European call options:

Filter

v0	vbar	vvol	rho	a
0.1100836	0.1888277	1.141432	-0.4564487	6.882538

Optimized constants for Heston model using historical data:

Filter

v0	vbar	vvol	rho	a
0.196	0.123	0.97	-0.567	0.98

Goldman Sachs Model - Pricing Code

Danny Ferdman, Hang (Joy) Che, Nima Nader and Tsung-Yen (Daniel) Ho

December 15, 2017

```
GS_Model <- function(S,
                     par,
                     rfRate,
                     maturity_Years,
                     coupon_rate,
                     creditSpread,
                     sigma,
                     timeIncrement = 1/12){

  rf = log(1 + rfRate)
  dt <- h <- timeIncrement #1/12 # Time Increment # Should be divisble
  #    by two to align to bond optionality
  maturity <- maturity_Years * 1 / dt# maturity in terms of number of evaluation periods
  rateDisc <- exp(rf) - 1

  conversionDisc <- rf
  parDisc <- log(1 + rateDisc + creditSpread)

  u <- exp((rf - 1/2 * sigma ^ 2) * h +  sigma * sqrt(h))
  d <- exp((rf - 1/2 * sigma ^ 2) * h -  sigma * sqrt(h))

  pUp = (exp(rf*h) - d)/(u-d)
  coupon = par * coupon_rate * h


  call_payoff = function(S_T, type = 'payoff'){
    if(type == 'payoff') {
      return(pmax(S_T, par + coupon))}

    else if(type == 'rate') {
      return(ifelse((S_T - (par + coupon)) > 0, conversionDisc * dt, parDisc * dt))}

    else {
      warning('Uknown type selection in the function!')}
  }

  # # Convertible Bond Features:
  # feat <- rbind('Time' = 1:maturity,
  #               'AbsCall' = c(NA, 115, 110, 105, NA),
  #               'Put' =     c(NA, NA, 120, NA, NA),
  #               'Coupon' = rep(coupon, maturity))

  # Stock Tree Building
  S_Tree <- sapply(X = 1:maturity,
                   FUN = function(x) c(S * u ^ (x:0) * d ^ (0:x),
                                       rep(NA, maturity - x))
```

```
)

payoffMat <- S_Tree * NA
rateMat <- S_Tree * NA

# Terminal Values:
payoffMat[, maturity] <- round(call_payoff(S_Tree[, maturity], type = 'payoff'), 2)

rateMat[, maturity]  <-  call_payoff(S_Tree[, maturity], type = 'rate')

# Backwards Iteration:
for (Time in (maturity -1):1){

  # C <- feat['Coupon', Time]
  C <- coupon

  rateMat[, Time] <- zoo::rollapply(
    data = rateMat[, Time + 1],
    width = 2,
    FUN = function(x) {x %*% (c(pUp, 1-pUp))},
    align = 'left',
    fill = NA)

  optimVal <-
    round(zoo::rollapply(
      # data = exp(-rateMat[, Time + 1] * h) * payoffMat[, Time + 1],
      # Discrete discounting to match the paper
      data = payoffMat[, Time + 1] / (1 + rateMat[, Time + 1]),
      width = 2,
      FUN = function(x) {x %*% (c(pUp, 1-pUp)) + C},
      align = 'left',
      fill = NA),
      2)

  # # Testing Callability and Puttability:
  # # Testing call first to get optimal behavior
  # # Call:
  # if (!is.na(feat['AbsCall', Time])) {
  #   strike <- feat['AbsCall', Time] + C
  #   rateMat[, Time] <- ifelse(optimVal > strike,
  #                             exp(parDisc) - 1,
  #                             rateMat[, Time])
  #   optimVal <- pmin(optimVal, strike)
  # }
  #
  # # Put:
  # if (!is.na(feat['Put', Time])) {
  #   strike <- feat['Put', Time] + C
  #   rateMat[, Time] <- ifelse(strike > optimVal,
  #                             exp(conversionDisc) - 1,
  #                             rateMat[, Time])
  #   optimVal <- pmax(optimVal, strike)
  # }
```

```
    # Compare to excercise Value
    rateMat[, Time] <- ifelse(S_Tree[, Time] > optimVal,
                              (exp(conversionDisc) - 1) * dt,
                              rateMat[, Time])
    optimVal <- pmax(optimVal, S_Tree[, Time])

    # Assign optimVal values to payoff Matrix
    payoffMat[, Time] <- round(optimVal, 2)

  }

  BondVal <- round(zoo::rollapply(
    # Discrete discounting to match the paper
    data = payoffMat[1:2, 1] / (1 + rateMat[1:2, 1]),
    width = 2,
    FUN = function(x) {x %*% (c(pUp, 1-pUp)) + C},
    align = 'left',
    fill = NA),
    2)[1]

  return(BondVal / par * 100)
}

# optim(par = .4, fn = function(x) abs(GS_Vol(x) - 1.40), method = 'Brent', lower = 0, upper = 1)
```

Goldman Sachs Model - Data Cleanup and Output Tables

Danny Ferdman, Hang (Joy) Che, Nima Nader and Tsung-Yen (Daniel) Ho

December 15, 2017

```
library(readxl)
library(data.table)
library(lubridate)
library(stringr)
library(zoo)
library(openxlsx)


# setwd('/Users/joyhangche/Dropbox/AFP/R Code')
setwd('C:/Users/Danny/Dropbox/Classes/AFP/R Code')
# setwd('C:/Users/DF/Dropbox/Classes/AFP/R Code')
source('GS_Model.R')

# Risk Free Rates:
rfRates <- fread('Interest Rates.csv',
                 stringsAsFactors = F,
                 header = T,
                 dec = '.',
                 sep = ',',
                 na.strings = 'N/A')
rfRates[, Date := mdy(Date)]
rfRates[, names(rfRates)[-1] := lapply(.SD[, -1], function(x) x / 100)]
rfRates[, '0Mo' := 0] # Setting this discount rates make life easier in subsequent steps
setnames(rfRates, gsub(' ', '', names(rfRates)))

bondUniv <- as.data.table(read_excel('Bond Universe.xlsx'))
setnames(bondUniv, gsub(' ', '', names(bondUniv)))

# Underlying (Equity) Data
stockData <- as.data.table(read_excel('Bond and Underlying Data.xlsx',
                                      sheet = 'Underlying Data',
                                      na = '#N/A N/A'))
tempNames <- names(stockData)
tempNames[str_detect(tempNames, 'X__')] <- ""
setnames(stockData, gsub('__1', '', tempNames))
rm('tempNames')
stockData <- stockData[, -which(is.na(unlist(stockData[1, ]))), with = F]
stocks <- (names(stockData)[which(names(stockData) != "")])

parList <- unlist(stockData[1, ])[1:(which(names(stockData) != "")[2] - 1)]

setnames(stockData, unlist(stockData[1, ]))
stockData <- stockData[-1, ]

stockList <- lapply(unique(stocks),
```

```
                    function(EQ){
                      EQ_Num <- which(stocks == EQ)[1]
                      begCol <- (1 + (EQ_Num - 1) * length(parList))
                      endCol <- begCol + length(parList) - 1
                      listElm <- stockData[, begCol : endCol]
                      listElm[, names(listElm) := lapply(listElm, as.numeric)]
                      volCols <- names(listElm)[str_detect(names(listElm), 'IMP_VOL|VOLATILITY')]
                      # browser()
                      listElm[, (volCols) := lapply(.SD, function(x) x / 100), .SDcols = volCols]
                      setnames(listElm, 'PX_LAST', 'StockPX')
                      listElm[, Name := EQ]
                      return(listElm)
                    })
names(stockList) <- unique(stocks)

stockDT <- rbindlist(stockList)
stockDT[, Date := as.Date(as.numeric(Date), origin = '1899-12-30')]

# Bond Data
bondData <- as.data.table(read_excel('Bond and Underlying Data.xlsx',
                                     sheet = 'Bond Data',
                                     na = '#N/A'))

tempNames <- names(bondData)
tempNames[str_detect(tempNames, 'X__')] <- ""
setnames(bondData, gsub('__1', '', tempNames))
rm('tempNames')
bondData <- bondData[, -which(is.na(unlist(bondData[1, ]))), with = F]

{bonds <- unique(names(bondData)[which(names(bondData) != "")])
bonds <- sapply(bonds,
                function(x){
                  end <- gregexpr(' ', x)[[1]][3] # Drop "Corp CUSIP"
                  substr(x, 1, end - 1)
                })}

bondParList <- unlist(bondData[1, ])[1:(which(names(bondData) != "")[2] - 1)]

setnames(bondData, unlist(bondData[1, ]))
bondData <- bondData[-1, ]

bondList <- lapply(bonds,
                    function(Bond){
                      Bond_Num <- which(bonds == Bond)[1]
                      begCol <- (1 + (Bond_Num - 1) * length(bondParList))
                      endCol <- begCol + length(bondParList) - 1
                      tryCatch(listElm <- bondData[, begCol : endCol],
                        error = function(e) {cat(Bond, '\n'); Sys.sleep(2)},
                        warning = function(w) {cat(Bond, '\n')})
                      # listElm[, Date := openxlsx::convertToDate(Date)]
                      listElm <- listElm[!is.na(Date)]
                      listElm[, Description := Bond]
                      listElm[, RSK_BB_IMPLIED_CDS_SPREAD :=
```

```
                                as.numeric(RSK_BB_IMPLIED_CDS_SPREAD) / 100 / 100]
                    listElm[, PX_VOLUME := as.numeric(PX_VOLUME)]
                    listElm[, PX_LAST := as.numeric(PX_LAST)]
                    setnames(listElm, 'PX_LAST', 'Bond_PX')
                    return(listElm)
                  })

# Remove Mising Bond Data:
{drop <- which(sapply(bondList,
              function(x) ifelse(x[1,1, with = F] == '#N/A Invalid Security', T, F))
       == T)
bondList[drop] <- NULL
if(length(bonds) > length(bondList)) {bonds <- bonds[-drop]}}

bondDT <- rbindlist(bondList)

trainBond <- bondUniv[Description %in% bonds, .(Description,
                                                Vol5D,
                                                UndTkr,
                                                ConvPx,
                                                Cpn = as.numeric(Cpn) / 100,
                                                EffMtyDate = lubridate::mdy(EffMtyDate))]

trainSet <- merge(x = trainBond,
                  y = bondDT[!is.na(RSK_BB_IMPLIED_CDS_SPREAD) &
                             !is.na(Bond_PX), ],
                  by = 'Description',
                  all = T)


trainSet[, Date := as.Date(as.numeric(Date), origin = '1899-12-30')]
trainSet[, TtoMaturity := as.numeric(EffMtyDate - Date) / 365]

trainSet <- merge(x = trainSet,
                  y = stockDT,
                  by.x = c('UndTkr', 'Date'),
                  by.y = c('Name', 'Date'),
                  all.x = T,
                  all.y = F)

# Missing dates are not material to be filled in...
# Fill in dates into rfRates table that are not in the data:
# missingDates <- data.table('Date' = trainSet[!(Date %in% rfRates$Date)]$Date)
# missingDates <- copy(rfRates)[1:nrow(missingDates)][, Date :=
#   trainSet[!(Date %in% rfRates$Date)]$Date]

# Adding in the correct interest rate for the period:
trainSet[, pseK := 1:.N]
# subSD interpolates interest rates. It was originally supposed to be
#   within a data table but I decided to move it out since I'm constantly using it.
subSD <- merge(trainSet[, .(Date, TtoMaturity, pseK)],
               y = rfRates,
               by = 'Date',
```

```
                all.x = T,
                all.y = F)
setorder(subSD, pseK)

matRF <- c('0Mo' = 0, sapply(names(rfRates)[-ncol(rfRates)][-1], # Risk Free Maturities Vector
                     function(x){
                       x <- gsub('Yr', '', x)
                       x <- gsub('Mo', '/12', x)
                       return(eval(parse(text = x)))
                     }))

subSD[, c('lowerYr', 'upperYr') :=
        list(matRF[sapply(TtoMaturity, function(x) which.max(x <= matRF) - 1)],
             matRF[sapply(TtoMaturity, function(x) which.max(x <= matRF))])
      ]


for (bounds in lapply(seq_along(matRF)[-1], function(x) matRF[c(x - 1, x)])){
  if (subSD[TtoMaturity %inrange% bounds, .N] != 0){
    subSD[TtoMaturity %inrange% bounds, c('lowerRF', 'upperRF') := {
    .SD[, names(bounds), with = F]
      }
    ]
  }
}

subSD[lowerYr == 0, lowerYr := .0001] # To shift the lower year a little bit to be able to
#    interpolate short maturity interest rate using a log model.

subSD[, RF_Rate := {unlist(
  lapply(transpose(.SD[, .(lowerYr, upperYr, lowerRF, upperRF, TtoMaturity)]),
         function(x){
           # cat(x[5], '\n')
           matrix(c(1, log(x[5])), nrow = 1) %*%
             solve(matrix(c(1, 1, log(x[2]), log(x[1])), ncol = 2)) %*%
             matrix(c(x[4], x[3]), ncol = 1)
           # browser()
         })
)}]

trainSet <- merge(x = trainSet, y = subSD[, .(pseK, RF_Rate)], by = 'pseK', all.x = T, all.y = F)

# Bloomberg has a data error for everything that has an Index for the underlying.
#   convertible bond for these rows are getting deleted.
trainSet <- trainSet[!stringr::str_detect(UndTkr, 'Index')]

# Adding PE and Price to Book:
getNewVars <- function(sheetName){
  newData <- as.data.table(read_excel('Bond and Underlying Data.xlsx',
                                      sheet = sheetName,
                                      na = '#N/A N/A'))
```

```
  setnames(newData, gsub(".*X__.*", "", names(newData)))
  stocks <- (names(newData)[which(names(newData) != "")])
  newData <- newData[, -which(is.na(unlist(newData[1, ]))), with = F]


  parList <- unlist(newData[1, ])[1:(which(names(newData) != "")[2] - 1)]

  setnames(newData, unlist(newData[1, ]))
  newData <- newData[-1, ]

  newList <- lapply(setNames(unique(stocks), unique(stocks)),
                    function(EQ){
                      EQ_Num <- which(stocks == EQ)[1]
                      begCol <- (1 + (EQ_Num - 1) * length(parList))
                      endCol <- begCol + length(parList) - 1
                      listElm <- newData[, begCol : endCol]
                      listElm[, names(listElm) := lapply(listElm, as.numeric)]
                      listElm[, Name := EQ]
                      listElm <- listElm[!is.na(Date)]
                      return(listElm)
                    })

  newDT <- rbindlist(newList)
  newDT[, Date := as.Date(as.numeric(Date), origin = '1899-12-30')]
  return(newDT)
}

for (i in list(c('PE Ratio', 'PE'), c('Price to Book', 'PtB'))) {assign(i[2], getNewVars(i[1]))}

# Merge the new factors into trainSet:
for (i in c('PE', 'PtB')){
  trainSet <- merge(x = trainSet,
                    y = get(i),
                    by.x = c('Date', 'UndTkr'),
                    by.y = c('Date', 'Name'),
                    all.x = T,
                    all.y = F)
}

setorder(trainSet, pseK)
setkey(trainSet, pseK)

# Add Industry to trainSet:
#   NB: due to Bloomberg excel add in issues at school the industry classificaiton is as
#   of the most recent date and not historical. There was nothing we could have done about it.
Industry <- as.data.table(read_excel('Industry Classification.xlsx'))
Industry <- Industry[, .SD[1], .(Description)] # To delete subsequent issues of the same security
trainSet <- merge(x = trainSet,
                  y = Industry[, .(Description, BICS)],
                  by = 'Description',
                  all.x = T,
                  all.y = F)
trainSet[, BICS := as.factor(BICS)]
```

```r
# Model Testing:
trainSet[!is.na(StockPX) &
           !is.na(ConvPx) &
           !is.na(Bond_PX) &
           !is.na(TtoMaturity) &
           !is.na(Cpn) &
           !is.na(RSK_BB_IMPLIED_CDS_SPREAD) &
           !is.na(RF_Rate) &
           !is.na(PX_VOLUME) &
           TtoMaturity < 1 &
           VOLUME_TOTAL_CALL > median(trainSet$VOLUME_TOTAL_CALL, na.rm = T),
        evalHere := 'modelTesting']

trainSet[evalHere == 'modelTesting', GS_PX :=
{
  dataSet <- transpose(.SD[,.(StockPX,
                              ConvPx,
                              RF_Rate,
                              TtoMaturity,
                              Cpn,
                              RSK_BB_IMPLIED_CDS_SPREAD,
                              `3MO_CALL_IMP_VOL`)])
  # Count <- 0
  # browser()
  unlist(lapply(dataSet,
                function(inputs){
                  inputs <- as.numeric(inputs)
                  # INPUTS <<- inputs
                  # str(inputs)
                  # browser()
                  # Count <<- Count + 1
                  # cat('+', Count, '\n')
                  GS_Model(S = inputs[1],
                               par = inputs[2] ,
                               rfRate = inputs[3],
                               maturity_Years = inputs[4],
                               coupon_rate = inputs[5],
                               creditSpread = inputs[6],
                               sigma = inputs[7])

                }))}
]

trainSet[evalHere == 'modelTesting', PX_Diff := GS_PX - Bond_PX]
hist(trainSet[evalHere=='modelTesting', PX_Diff/Bond_PX],
      breaks = 200,
      col = 'blue',
      main = 'GS Model PX Diff \nDistribution',
      xlab = 'Price (PX) Difference')
trainSet[evalHere=='modelTesting', .SD[abs(PX_Diff/Bond_PX)<.02, .N] /.N]
# axis(1, at=seq(0,1, by=0.1), labels=seq(0,1, by=0.1))

trainSet[!is.na(StockPX) &
```

```
            !is.na(ConvPx) &
            !is.na(Bond_PX) &
            !is.na(TtoMaturity) &
            !is.na(Cpn) &
            !is.na(RSK_BB_IMPLIED_CDS_SPREAD) &
            !is.na(RF_Rate) &
            !is.na(PX_VOLUME) &
            TtoMaturity > 1 &
            VOLUME_TOTAL_CALL > median(trainSet$VOLUME_TOTAL_CALL, na.rm = T),
          evalHere := {
            tryCatch(rev(c(rep('T1', 5), rep(NA, max(.N - 5, 0)))[1:.N]),
                     error = function(e) browser())
            },
          .(Description)]

# The below code is to extract implied vol for the period to maturity as forecasted by  the GS
#   model. NB: it takes a few hours to run. Load trainSet-Final.RDS which includes solvedVol.
trainSet[evalHere == 'T1' & pseK == 17081, solvedVol :=
          {#dataSet <- transpose(.SD[!is.na(StockPX)])
          dataSet <- transpose(.SD[,.(StockPX,
                                      ConvPx,
                                      RF_Rate,
                                      TtoMaturity,
                                      Cpn,
                                      RSK_BB_IMPLIED_CDS_SPREAD,
                                      Bond_PX)])
          Count <- 0
          unlist(lapply(dataSet,
                 function(inputs){
                   inputs <- as.numeric(inputs)
                   # INPUTS <<- inputs
                   # str(inputs)
                   # browser()
                   Sol <- optim(par = .4,
                         fn = function(vol) abs(GS_Model(S = inputs[1],
                                                         par = inputs[2] ,
                                                         rfRate = inputs[3],
                                                         maturity_Years = inputs[4] ,
                                                         coupon_rate = inputs[5],
                                                         creditSpread = inputs[6],
                                                         sigma = vol) -
                                                  inputs[7]), method = 'Brent', lower = 0, upper = 1)
                   Count <<- Count + 1
                   cat('+', Count, ' ', Sol$par, '\n')
                   # browser()
                   Sol$par
                 }))}
            ]

tbackup <- copy(trainSet)

# Forecast Options Implied Volatility Haircut-
#   Linear Model Results:
```

```
trainSet[!is.na(CUR_MKT_CAP) &
           !is.na(`3MO_CALL_IMP_VOL`) &
           solvedVol %inrange% c(.15, .6) & # The narrow range for vol is to discard
           #   bonds that most likely embedded feature that muddy the results for
           #   convertible bonds without embedded options
           !is.na(PX_TO_BOOK_RATIO) &
           !is.na(PE_RATIO) &
           !is.na(VOLATILITY_90D) &
           !is.na(BICS) &
           TtoMaturity > 1,
         {
             moneyness <- (StockPX - ConvPx)
             lmOut <-
               lm((solvedVol / (`3MO_CALL_IMP_VOL`)) ~
                    log(CUR_MKT_CAP) +
                    {sign(moneyness) * (abs(moneyness) / StockPX)} +
                    TtoMaturity +
                    (PE_RATIO) +
                    (PX_TO_BOOK_RATIO) +
                    (VOLATILITY_90D) +
                    BICS)
             # stargazer function makes the lm printout look nicer:
             # (stargazer::stargazer(lmOut,
             #                       type = 'text',
             #                       report = 'vc*p',
             #                       covariate.labels = c(NA,
             #                         'sign(moneyness) * (abs(moneyness) / StockPX)',
             #                         NA, NA, NA, NA, NA)))
             sink('RegressionResults.txt')
             print(summary(lmOut))
             sink()
           }
         ]


# Test the distribution using fitted values from the linear regression:
trainSet[!is.na(CUR_MKT_CAP) &
           !is.na(`3MO_CALL_IMP_VOL`) &
           solvedVol %inrange% c(.15, .6) & # The narrow range for vol is to discard
           #   bonds that most likely embedded feature that muddy the results for
           #   convertible bonds without embedded options
           !is.na(PX_TO_BOOK_RATIO) &
           !is.na(PE_RATIO) &
           !is.na(VOLATILITY_90D) &
           !is.na(BICS) &
           TtoMaturity > 1,
         fittedVol :=
             {
             moneyness <- (StockPX - ConvPx)
             lm((solvedVol / (`3MO_CALL_IMP_VOL`)) ~
                  log(CUR_MKT_CAP) +
                  {sign(moneyness) * (abs(moneyness) / StockPX)} +
                  TtoMaturity +
```

```
                  PE_RATIO +
                  PX_TO_BOOK_RATIO +
                  VOLATILITY_90D +
                  BICS)$fitted * `3MO_CALL_IMP_VOL`
          }
        ]


trainSet[!is.na(fittedVol), fittedPX :=
  {
   dataSet <- transpose(.SD[,.(StockPX,
                               ConvPx,
                               RF_Rate,
                               TtoMaturity,
                               Cpn,
                               RSK_BB_IMPLIED_CDS_SPREAD,
                               fittedVol)])
   Count <- 0
   unlist((lapply(dataSet,
          function(inputs){
            inputs <- as.numeric(inputs)
            # INPUTS <<- inputs
            # str(inputs)
            # browser()
            Sol <- GS_Model(S = inputs[1],
              par = inputs[2] ,
              rfRate = inputs[3],
              maturity_Years = inputs[4] ,
              coupon_rate = inputs[5],
              creditSpread = inputs[6],
              sigma = inputs[7])
           Count <<- Count + 1
           cat('+', Count, '\n')
           Sol})))
   }
]

trainSet[, fittedPX_Diff := Bond_PX - fittedPX]

jpeg('Forecasted Vol Hist.jpg')
hist(trainSet[Description != 'ENDP 1.5 07/15/18', fittedPX_Diff], # Exclude ONE outlier observation
     60,
     col = 'lightgreen',
     main = 'GS Model Price \nDifference Distribution',
     xlab = 'Fitted Price (PX) Difference',
     cex.lab = 1.25,
     cex.main = 1.5)
dev.off()

# How many of the observations lie within +/- 5 points:
trainSet[fittedPX_Diff > -20, .SD[fittedPX_Diff %inrange% c(-2,2), .N] / .N]

# Testing model sensitivities to different inputs:
```

```
# Mid point parameter specification for analysis:
initS <- 100; initPar <- 360/245*100; initRF <- .05; initMat <- 5;
initC <- .1; initSpread <- .013; initSig <- .2

# Create a list for all the values to be passed into the GS Model:
shockTable <- list(
  'S' = seq(initS * .8, initS * 1.2, initS * .01),
  'par' = seq(initPar * .8, initPar * 1.2, initPar * .01),
  'rfRate' = seq(initRF * .8, initRF * 1.2, initRF * .01),
  'maturity_Years' = round(seq(initMat - 20*1/12, initMat + 20*1/12, 1/12),4),
  'coupon_rate' = seq(initC * .8, initC * 1.2, initC * .01),
  'creditSpread' = seq(initSpread * .8, initSpread * 1.2, initSpread * .01),
  'sigma' = seq(initSig * .8, initSig * 1.2, initSig * .01))

# Create a table for all the values to be passed into the GS Model from the shock list:
senseTable <- rbindlist(lapply(seq_along(shockTable),
                     function(x){
                       DT <- setNames(data.table(unlist(shockTable[x])), names(shockTable[x]))
                       DT[, names(shockTable[-x]) := list(initS,
                                                          initPar,
                                                          initRF,
                                                          initMat,
                                                          initC,
                                                          initSpread,
                                                          initSig)[-x]]
                       setcolorder(DT, names(shockTable))
                       DT[, shockVar := names(shockTable)[x]]
                       return(DT)
                     }))

# Extract prices from the model:
senseTable[, PX :=
  {
  count <- 0
  unlist(lapply(transpose(.SD[, names(shockTable), with = F]),
                          function(X){
                            inputs <- unlist(X)
                            count <<- count + 1
                            cat(count, '\n')
                            tryCatch(GS_Model(S = inputs[1],
                                     par = inputs[2] ,
                                     rfRate = inputs[3],
                                     maturity_Years = inputs[4] ,
                                     coupon_rate = inputs[5],
                                     creditSpread = inputs[6],
                                     sigma = inputs[7]),
                                     error = function(e) {browser(); return(NA)})
                          }
                        )
                       )
    }
  ]
```

```r
# Visually Checking the Residuals:
# apply(trainSet[fittedPX_Diff > -20, {
#   moneyness <- (StockPX - ConvPx)
#   .(log(CUR_MKT_CAP),
#   {sign(moneyness) * (abs(moneyness) / StockPX)},
#   TtoMaturity,
#   PE_RATIO,
#   PX_TO_BOOK_RATIO,
#   VOLATILITY_90D)}],
# MARGIN = 2,
# function(X) {
#   plot(y = trainSet[fittedPX_Diff > -20, fittedPX_Diff],
#        x = X)
# })

# Plot the price sensitivities with respect to the variables:
par(mfrow = c(2,2))
for (i in seq_along(shockTable)[c(1,2,4,7)]){
  plot(senseTable[shockVar == names(shockTable)[i], c(names(shockTable)[i], 'PX'), with = F],
       type = 'o',
       main = paste('Sensitivity Analysis w.r.t',
                    ifelse(names(shockTable)[i] =='par',
                           'Conversion Price',
                           names(shockTable)[i])),
       ylab = 'Bond Price',
       xlab = ifelse(names(shockTable)[i] =='par',
                     'Conversion Price',
                     names(shockTable)[i]))
}
dev.off()
```

Stochastic Volatility Model

```
#Convertible Bond Pricing:

######################################################################

############################## Data ##############################

library(readxl)
```

```
## Warning: package 'readxl' was built under R version 3.3.3
```

```
Data <- read_excel("C:/Users/Nima Nader/Desktop/AFP NIMA/Convertible Bond Pricing/InputData.xlsx")

######################################################################

###################### Heston Model Data ######################

# v0: initial volatility
V0<-as.numeric(Data[1,2])
# vbar: long-term variance mean
vbar<-as.numeric(Data[2,2])
# vvol: volatility of the variance process
vvol<-as.numeric(Data[3,2])
# rho: corrolation between Weiner Process for stock and volatility
rho<-as.numeric(Data[4,2])
# a: variance mean revrsion
a<-as.numeric(Data[5,2])
# so: Initial Stock Price
S0<-as.numeric(Data[6,2])

######################################################################

######################### CIR Model Data #########################

# r0: risk free rate at time 0
r0<-as.numeric(Data[1,4])
# kk: mean recersion speed
kk<-as.numeric(Data[2,4])
# sigma: volatility
sigma<-as.numeric(Data[3,4])
# rbar: long term mean
rbar<-as.numeric(Data[4,4])

######################################################################

############################ Bond Data ############################

# TT: Maturity Time:
TT<-as.numeric(Data[1,6])
# CR: coupon rate:
CR<-as.numeric(Data[2,6])
#tt1: initial coupon date:
```

```
tt1<-as.numeric(Data[3,6])
# fc: frequent of coupon payment
fc<-as.numeric(Data[4,6])
# t1: conversion Starting Time
t1<-as.numeric(Data[5,6])
# t2: conversion ending Time
t2<-as.numeric(Data[6,6])
# Cratio: conversion Ratio
Cratio<-as.numeric(Data[7,6])
#Par: Par amount
Par<-as.numeric(Data[8,6])
#IPT: init Premium
IPT<-as.numeric(Data[9,6])

######################################################################

################## Monte Carlo Simulation Data ###################

# N: number of simulations
N<-100

######################################################################

###################### Discount Function ########################

Discount <- function(rt,tt,TT,kk,sigma,rbar){

  h<-sqrt(kk^2+2*sigma^2)
  AtT <- ((2*h*exp((kk+h)*(TT-tt)/2))/(2*h+(kk+h)*(exp((TT-tt)*h)-1)))^(2*kk*rbar/(sigma^2))
  BtT <- (2*(exp((TT-tt)*h)-1))/(2*h+(kk+h)*(exp((TT-tt)*h)-1))

  P <- AtT*exp(-BtT*rt)

  return(P)

}

######################################################################

################## Discount to time zero #####################

PoftT<-c()
j<-1

for (i in seq(from=0,to=TT,by=1/252)){

  PoftT[j]<-Discount(r0,0,i,kk,sigma,rbar)
  j<-j+1

}

#plot(PoftT)
######################################################################
```

```
#Check

lll<--1*log(PoftT)
riskfreeeeeee<-c()
hinim<-seq(from=0,to=TT,by=1/252)
riskfreeeeeee[1]<-r0
for(i in 2:length(lll)){

  riskfreeeeeee[i]<-lll[i]/hinim[i]


}


#################### generate stock paths ####################

StockPath<-matrix(data=NA,nrow=N,ncol=TT*252+1)
h<-1/252

for(i in 1:N){

  Volatility<-c()
  riskfreerate<-c()
  StockPath[i,1]<-S0
  Volatility[1]<-V0
  riskfreerate[1]<-r0

  for(j in 2:(TT*252+1)){

    dw1<-rnorm(1)
    dw2<-rnorm(1)
    dw3<-rnorm(1)

    Volatility[j] <- Volatility[j-1] + a*(vbar-Volatility[j-1])*h+
      vvol*sqrt(abs(Volatility[j-1]))*sqrt(h)*dw1
    #riskfreerate[j] <- riskfreerate[j-1] + kk*(rbar-riskfreerate[j-1])*h +
    #sigma*sqrt(max(0,riskfreerate[j-1]))*sqrt(h)*dw2
    StockPath[i,j] <- StockPath[i,j-1] + riskfreeeeeee[j-1]*StockPath[i,j-1]*h+sqrt(abs(Volatility[j-1]

    #riskfreerate[j-1]
  }

}

#################################################################

#################### generate Coupon Path ####################

CouponPayments<-matrix(data=0,nrow=1,ncol=TT*252+1)

while(tt1<=TT){

  CouponPayments[1,tt1*252+1]<-Par*CR
```

```
  tt1<-tt1+fc

}

CouponPayments[1,TT*252+1]<-CouponPayments[1,TT*252+1]+Par

#####################################################################

################## Pricing Convertible Bond ##################

StockPath<-StockPath*Cratio
discountedStockPath<-StockPath
discountedCouponPayments<-CouponPayments
discountedCouponPayments[1,]<-discountedCouponPayments[1,]*PoftT

for(i in 1:N){

  discountedStockPath[i,]<-discountedStockPath[i,]*PoftT

}

discountedStockPath[,seq(from=1,to=t1*252+1,by=1)]<-0
discountedStockPath[,seq(from=t2*252+1,to=TT*252+1,by=1)]<-0

Price<-c()

for (i in 1:N){

  Price[i]<-max(discountedCouponPayments[1,TT*252+1],discountedStockPath[i,TT*252+1])

  for(j in seq(from=TT*252,to=1,by=-1)){

    Price[i]<-max(Price[i]+discountedCouponPayments[1,j],discountedStockPath[i,j])

  }

}

FinalPrice<-100*mean(Price)/Par
#FinalPrice

########################### END ################################
```

Optimizing CIR Model

```
# Optimizing CIR Model:

#####################################################################

############################## Data ##############################

library(readxl)
```

```
## Warning: package 'readxl' was built under R version 3.3.3
```

```
CMT <- read_excel("C:/Users/Nima Nader/Desktop/AFP NIMA/CIR Model/CMT.xlsx")
CMT[,2:9]<-as.matrix(CMT[,2:9]/100)
CMT<-CMT[1:10,1:9]

#####################################################################

########################## Variables ##############################

# rt: risk free rate at time t
# rbar: long term mean
# sigma: volatility
# kk: mean recersion speed
# tt: initial time
# TT: Maturity time

#####################################################################

###################### Discount Function ######################

Discount <- function(rt,tt,TT,kk,sigma,rbar){

  h<-sqrt(kk^2+2*sigma^2)
  AtT <- ((2*h*exp((kk+h)*(TT-tt)/2))/(2*h+(kk+h)*(exp((TT-tt)*h)-1)))^(2*kk*rbar/(sigma^2))
  BtT <- (2*(exp((TT-tt)*h)-1))/(2*h+(kk+h)*(exp((TT-tt)*h)-1))

  P <- AtT*exp(-BtT*rt)

  return(P)

}

#####################################################################

################## rt calculate using Spot Rate ################

rt <- function(YTM,tt,TT,kk,sigma,rbar){

  h<-sqrt(kk^2+2*sigma^2)
  AtT <- ((2*h*exp((kk+h)*(TT-tt)/2))/(2*h+(kk+h)*(exp((TT-tt)*h)-1)))^(2*kk*rbar/(sigma^2))
  BtT <- (2*(exp((TT-tt)*h)-1))/(2*h+(kk+h)*(exp((TT-tt)*h)-1))
```

```
  rt <- (YTM*(TT-tt)+log(AtT))/BtT

  return(rt)

}

#####################################################################

################## Spot Rate calculate using rt ################

YTM <- function(rtt,tt,TT,kk,sigma,rbar){

  h<-sqrt(kk^2+2*sigma^2)
  AtT <- ((2*h*exp((kk+h)*(TT-tt)/2))/(2*h+(kk+h)*(exp((TT-tt)*h)-1)))^(2*kk*rbar/(sigma^2))
  BtT <- (2*(exp((TT-tt)*h)-1))/(2*h+(kk+h)*(exp((TT-tt)*h)-1))

  YTM <- (-log(AtT)+BtT*rtt)/(TT-tt)

  return(YTM)

}

#####################################################################

######################### Cost Function ############################

Cost <- function(X){

  costis <- c()

  for(j in 1:length(CMT$Date)){

    costis[j] <- 0
    tt <- 0
    TT <- c(1/12,1/4,1/2,1,2,3,5,7)
    r0 <- rt(CMT[j,2],tt,TT[1],X[1],X[2],X[3])

    for(i in 2:9){

      costis[j] <- (as.numeric(YTM(r0,tt,TT[i-1],X[1],X[2],X[3])) - as.numeric(CMT[j,i]))^2 + costis[j]

    }

  }

  return(sum(costis))

}

#####################################################################

######################### Optimization ############################
```

```r
#x0 <- c(kk,sigma,rbar)

x0 <- c(0.25,0.5,0.2)
opt<-optim(par=x0,fn=Cost)

kk<-as.numeric(opt$par[1])
sigma<-as.numeric(opt$par[2])
rbar<-as.numeric(opt$par[3])

CIR_SOL<-matrix(data=c(as.numeric(opt$par[1]),as.numeric(opt$par[2]),as.numeric(opt$par[3])),nrow=1)
colnames(CIR_SOL)<-c('kk','sigma','rbar')

#CIR_SOL

######################################################################

####################### Test the Result ##########################

test<-matrix(ncol=9,nrow=3)
colnames(test)<-colnames(CMT)
rownames(test)<-c("actual",'CIR Estimate',"% error")

b<-length(CMT$Date)
test[1,] <- as.matrix(CMT[b,])

rtt<-rt(CMT[b,2],0,1/12,kk,sigma,rbar)
TT<-c(1/12,1/4,1/2,1,2,3,5,7)
for(i in 1:length(TT)){
  test[2,i+1] <- as.numeric(YTM(rtt,0,TT[i],kk,sigma,rbar))
}

for(i in 1:length(TT)){
  test[3,i+1] <- abs(100*(as.numeric(test[2,i+1])-as.numeric(test[1,i+1]))/as.numeric(test[1,i+1]))
}

r0 <- rt(CMT[b,2],0,1/12,kk,sigma,rbar)

#######################      END      ##########################
```

optimize Heston model using European call option

```
#Heston Model using call:

#####################################################################

########################## Call Data ###########################

library(optimx)
```

```
## Warning: package 'optimx' was built under R version 3.3.3
```

```
library(readxl)
```

```
## Warning: package 'readxl' was built under R version 3.3.3
```

```
CallPrices <- read_excel("C:/Users/Nima Nader/Desktop/AFP NIMA/Heston Model/CallPrices.xlsx")

#####################################################################

########################## Variables ###########################

# so: Initial Stock Price
# v0: initial volatility
# vbar: long-term variance mean
# a: variance mean revrsion
# vvol: volatility of the variance process
# r: risk free rate
# rho: corrolation between Weiner Process for stock and volatility
# w: points at which to evaluate the function
# t: time to maturity
# k: option strike
i<-0+1i

################ Heston Characteristic Function #################

chfun_heston <- function(s0,v0,vbar,a,vvol,r,rho,t,w){

  alpha <- -w^2/2-i*w/2
  beta <- a -rho*vvol*w*i
  gamma <- vvol^2/2
  h <- sqrt(beta^2-4*alpha*gamma)
  rplus <- (beta+h)/(vvol^2)
  rminus <- (beta-h)/(vvol^2)
  g <- rminus/rplus
  C <- a*(rminus*t-(2/(vvol^2))*log((1-g*exp(-h*t))/(1-g)))
  D <- rminus*(1-exp(-h*t))/(1-g*exp(-h*t))
  f<-exp(C*vbar+D*v0+i*w*log(s0*exp(r*t)))

  return(f)
}

#####################################################################
```

```
############### Call Price using Heston Model #################

call_heston_cf <- function(s0,v0,vbar,a,vvol,r,rho,t,k){

  int <- function(w){

    inta <- Re(exp(-i*w*log(k))*chfun_heston(s0,v0,vbar,a,vvol,r,rho,t,w-i)/
                 (i*w*chfun_heston(s0,v0,vbar,a,vvol,r,rho,t,-i)))

    return(as.numeric(inta))

  }

  int1 <- integrate(int,lower=0,upper=Inf,stop.on.error = FALSE)
  pi1 <- 0.5 + as.numeric(int1[1])/pi

  intb <- function(w){

    intc <- Re(exp(-i*w*log(k))*chfun_heston(s0,v0,vbar,a,vvol,r,rho,t,w)/(i*w))

    return(intc)

  }

  int2 <- integrate(intb,lower=0,upper=Inf,stop.on.error = FALSE)
  pi2 <- 0.5 + as.numeric(int2[1])/pi

  CALLPRICE <- s0*pi1-exp(-r*t)*k*pi2

  return(CALLPRICE)

}

################################################################

############## Cost Function for Optimization #################

Cost <-function(x){

  costis <- c()

  for (j in 1:length(CallPrices$Spot)){

    costis[j] <- CallPrices$Mid[j]-call_heston_cf(CallPrices$Spot[j],x[1],
                                                  x[2],(x[5]+x[3]^2)/(2*x[2]),x[3],
                                                  CallPrices$`Interest rate`[j],x[4],
                                                  CallPrices$Maturity[j],CallPrices$Strike[j])

  }

  return(sum(costis^2))

}
```

```
#####################################################################

######################### Optimization ##############################

#x0 <- c("v0","vbar","vvol","rho","2*a*vbar-vvol^2")

x0 <- c(0.1,0.1,1,-0.5,1)
lb <- c(0,0,0,-1,0)
ub <- c(1,1,5,1,10)

opt<-optimx(par=x0,fn=Cost,hessian=T,lower=lb,upper=ub,method="bobyqa")

Heston_sol<-matrix(data=c(as.numeric(opt$p1),as.numeric(opt$p2),as.numeric(opt$p3),
                          as.numeric(opt$p4),as.numeric((opt[5]+opt[3]^2)/(2*opt[2]))),nrow=1)
colnames(Heston_sol)<-c("v0","vbar","vvol","rho","a")

#Heston_sol

#####################################################################

######################### Test the Result ###########################

TestedCallPrices<-c()

for(j in 1:length(CallPrices$Spot)){

  TestedCallPrices[j] <- call_heston_cf(CallPrices$Spot[j],Heston_sol[1],
                                        Heston_sol[2],Heston_sol[5],Heston_sol[3],
                                        CallPrices$`Interest rate`[j],Heston_sol[4],
                                        CallPrices$Maturity[j],CallPrices$Strike[j])

}

Test<-matrix(data=NA,ncol=6,nrow=length(CallPrices$Spot))
colnames(Test)<-c("Mid","Bid","Ask","Calc","Within-Bid-Ask?","% Error")

Test[,1]<-CallPrices$Mid
Test[,2]<-CallPrices$Bid
Test[,3]<-CallPrices$Ask
Test[,4]<-round(TestedCallPrices,digits=2)

for(j in 1:length(CallPrices$Spot)){

  if((2*abs(as.numeric(Test[j,4])-as.numeric(Test[j,1])))<=(as.numeric(Test[j,3])-
                                                            as.numeric(Test[j,2]))){

    Test[j,5]<-"Yes"
    Test[j,6]<-0

  }else{

    Test[j,5]<-"No"
    Test[j,6]<-round(100*(abs(as.numeric(Test[j,4])-
```

```
                                as.numeric(Test[j,1]))/as.numeric(Test[j,1])),digits=2)

    }

}

############################### END ################################
```

optimize Heston model using historical data

```
# Historic Data Optimization

#####################################################################

############################## Data ##############################

library(optimx)
```

```
## Warning: package 'optimx' was built under R version 3.3.3
```

```
library(data.table)
```

```
## Warning: package 'data.table' was built under R version 3.3.2
```

```
CMT <- data.table(read.csv("C:/Users/Nima Nader/Desktop/AFP NIMA/Optimization Using Historic Data/CMT.cs
StockData <- data.table(read.csv("C:/Users/Nima Nader/Desktop/AFP NIMA/Optimization Using Historic Data/

#####################################################################

######################### Merging Data ##########################

StockData<-StockData[,c(1,2)]
CMT<-CMT[,c(1,2)]

StockData<-StockData[,onemonth:=shift(close,21,type="lead")]
#StockData[,Threemonth:=shift(close,63,type="lead")]

MergedData<-merge(x=StockData,y=CMT,by.x="date",by.y="Date",all.x=TRUE,sort = FALSE)

a<-which(is.na(MergedData$onemonth))
b<-which(is.na(MergedData$X1.Mo))

if(length(a!=0)){

  MergedData<-MergedData[-a,]

}
if(length(b!=0)){

  MergedData<-MergedData[-b,]

}

#####################################################################

#################### StockPrice Function  ####################

StuckPrice<-function(s0,r,V0,vbar,vvol,rho,a){

h<-1/12

dw1<-rnorm(1)
```

```
dw2<-rnorm(1)

Volatility <- V0 + a*(vbar-V0)*h+vvol*sqrt(abs(V0))*sqrt(h)*dw1
Stockprice <- s0 + r*s0*h + sqrt(abs(Volatility))*s0*sqrt(h)*(rho*dw1+sqrt(1-rho^2)*dw2)

return(Stockprice)

}

#####################################################################

############## Cost Function for Optimization #################


Cost <-function(x){

  costis <- c()

  for (j in 1:length(MergedData$date)){

    costis[j] <- as.numeric(MergedData$onemonth[j])-
      as.numeric(StuckPrice(as.numeric(MergedData[j,2]),
                            as.numeric(MergedData[j,3]),x[1],x[2],x[3],x[4],x[5]))

  }

  return(sum(costis^2))

}

#####################################################################

####################### Optimization ############################

#x0 <- c("v0","vbar","vvol","rho","a")
N=10
Heston_sol<-matrix(data=NA,ncol=5,nrow=N)
colnames(Heston_sol)<-c("v0","vbar","vvol","rho","a")

x0 <- c(0.1,0.1,1,-0.5,1)

for(i in 1:N){

  opt<-optim(par=x0,fn=Cost)

  Solution<-matrix(data=c(as.numeric(opt$par)[1],as.numeric(opt$par)[2],as.numeric(opt$par)[3],
                          as.numeric(opt$par)[4],as.numeric(opt$par)[5]),nrow=1)

  Heston_sol[i,]<-Solution

}

Heston_solution<-matrix(data=c(mean(Heston_sol[,1]),mean(Heston_sol[,2]),mean(Heston_sol[,3]),
```

```
                                mean(Heston_sol[,4]),mean(Heston_sol[,5])),ncol=5,nrow=1)
colnames(Heston_solution)<-c("v0","vbar","vvol","rho","a")

#####################################################################
```