# Option Pricing under a Non-Parametric Distribution

Himanshu Babbar
University of North Carolina, Charlotte

## 1 Abstract

Most option pricing models assume a parametrized distribution for the prices of the underlying, and use it for pricing the options. However, the assumed distribution may not accurately model the price movements of the underlying, and hence, result in incorrect option prices. This paper presents a market-based valuation methodology to price options using a non-parametric distribution extracted from market data.

## 2 Introduction

The price of a European Call option, under the risk-neutral measure $Q$, is given as shown in Figure 1:

$$E^Q[e^{-rT}(S_T - K)^+] = \int_0^\infty (1 - F_{S_T}(x))\mathrm{d}x$$

where,
$r$ is the risk-free rate of interest
$T$ is the time to maturity of the option
$S_T$ is the price of the underlying at maturity
$K$ is the strike price of the option
$F_{S_T}(\cdot)$ is the CDF of $S_T$ under the risk neutral measure

If $S_0$ is considered to be the initial price of the underlying, then the above formula can be written as:

$$S_0 e^{-rT} E^Q\left[\left(\frac{S_T}{S_0} - \frac{K}{S_0}\right)\right] = S_0 e^{-rT} \int_{K/S_0}^\infty (1 - F_{S_T/S_0}(x))\mathrm{d}x$$

The CDF of $S_T/S_0$, $F_{S_T/S_0}(\cdot)$, can be estimated empirically using data obtained from the market.

## 3 Methodology

The PDF and CDF of a standard normal distribution is given as:

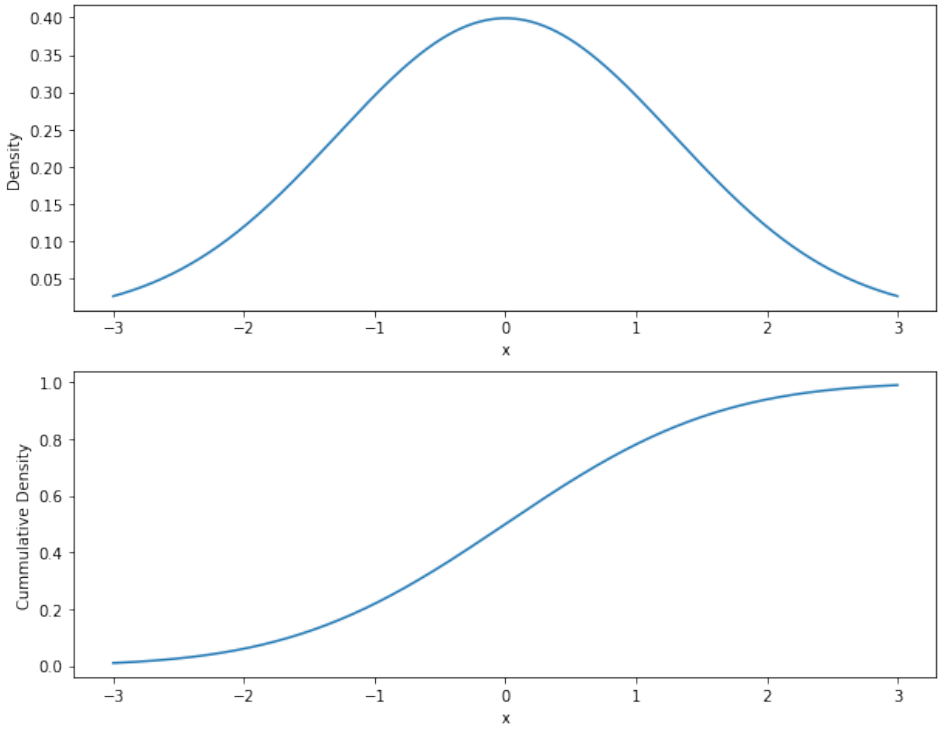


Figure 1: PDF and CDF of a Standard Normal Distribution

The curve for the CDF of a distribution can be modelled using a logistic function. A generalized logistic function is given as:

$$f(x) = A + \frac{K - A}{(C + Qe^{-B(x-M)})^{1/\nu}}$$

where,
$A$ is the lower asymptote
$K$ is the upper asymptote

$B$ is the growth rate
$\nu > 0$ affects near which asymptote maximum growth occurs
$Q$ is related to the value $f(0)$
$C$ mostly takes a value of 1
$M$ can be considered to be the starting time

Using $A = 0; K = 1; B = 2; \nu = 0.5; Q = 1; C = 1$ *and* $M = 1$, we can approximately model the CDF of a standard normal distribution as shown in Figure 2.

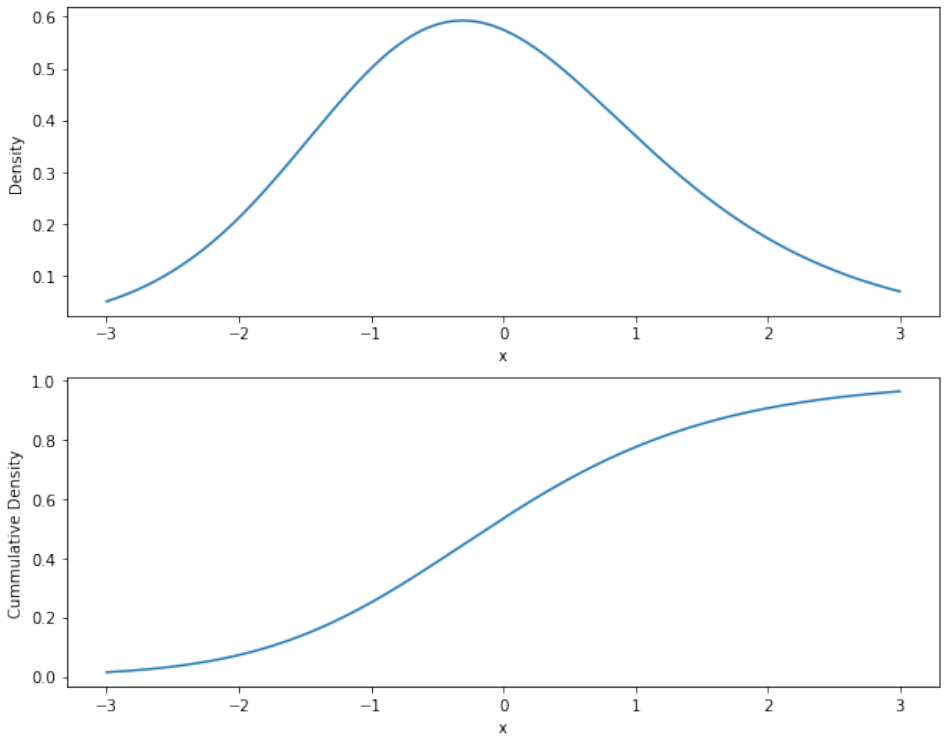


Figure 2: PDF and CDF modelled using a Generalized Logistic Function

The logistic function selected for modelling the CDF is as follows:

$$f(x) = \frac{1}{(1 + Qe^{-B(x-1)})^{1/\nu}}$$

where the parameters $Q, B$ *and* $\nu$ can be calibrated using market data.

# 4 Calibration Process

## 4.1 Data Used

The data used for the purpose of calibration are the SPX options with a single maturity for all the strikes within 2% of the current index level.

## 4.2 Process

The process, as suggested in [1], involves finding the best parameters that minimizes the Root Mean-Squares Error (RMSE) between the observed call prices and the ones calculated by the model. Thus, the aim is to solve the following

$$\min_{Q,B,\nu} \sqrt{\frac{1}{N}\sum_{n=1}^{N}(\hat{C}_n - C_n(Q,B,\nu))^2}$$

where, $\hat{C}$ are the observed market prices and $C(Q,B,\nu)$ are the prices calculated by the model based in the parameters $Q, B \; and \; \nu$.

## 4.3 Result

Using the above data, the values obtained for the parameters are as follows:

$$Q = 7.6968565,\ B = 157.57587127,\ \nu = 2.34054796$$

which resulted in a minimum RMSE of **0.043487**

Figure 3 shows how the prices calculated by the model fit the observed market prices. Figure 4 shows the PDF and CDF of the distribution extracted from the market data.

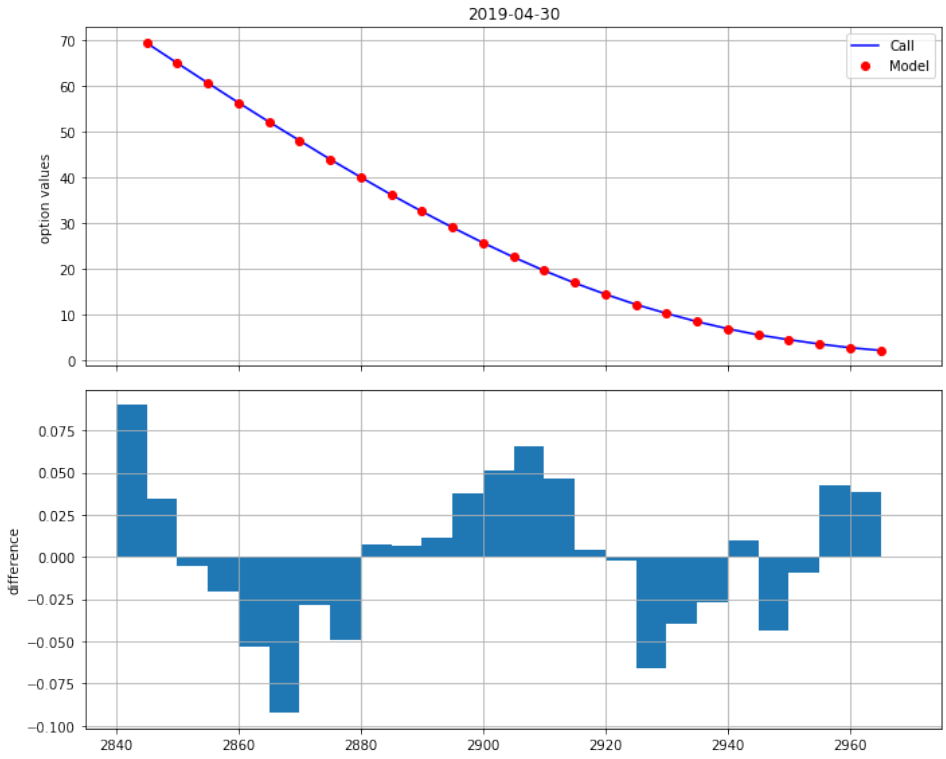


Figure 3: Calibration Result.

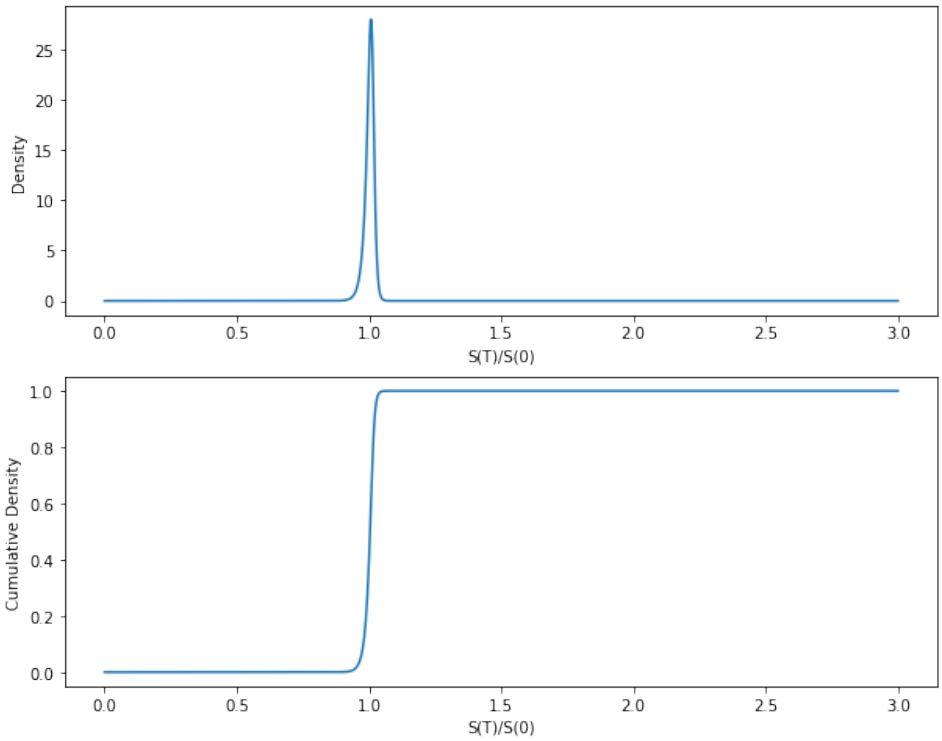


Figure 4: PDF and CDF Extracted from the market data

### 4.4 A Comparison

Table 1 is a comparison of the model presented in this paper with Merton's Jump Diffusion model(1976) ($M^{76}$) [1] as evaluated using the FFT approach proposed by Carr and Madan (1999) [1].

Table 1: Comparison.

| | **Non-Parametric Model** | **Merton 76** |
|---|---|---|
| Number of Parameters | 3 | 4 |
| RMSE | 0.04349 | 2.53701 |
| Computation Time (seconds) | 0.00159 | 0.00747 |

## 5 Conclusion

This paper developed an option pricing model using a non-parametric distribution. The results show that this model provides better calibration results than that achieved with $M^{76}$ model when calibrated using the FFT approach as proposed by Carr and Madan (1999) with a RMSE of 0.04349 V/s 2.53701. Moreover, the non-parametric model presented in this paper calibrates 3 parameters in comparison to 4 parameters in $M^{76}$ model. Also, the computation time for the non-parametric model is 0.00159 V/s 0.00747 for $M^{76}$ model when computed using the FFT approach. The non-parametric model, similar to $M^{76}$ model, however, do not have enough degrees of freedom to calibrate for multiple maturities.

## References

[1] Y. Hilpisch, *Derivatives Analytics with Python - Data Analysis, Models, Simulation, Calibration and Hedging.* John Wiley & Sons, 2015.

# A Python Code

```
# Get the Data
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as sop
from scipy.integrate import quad

import warnings
warnings.filterwarnings('ignore')

r = 0.005

optData = pd.read_csv('..'+os.sep+'data'+os.sep+'projectData.csv')
optData['Date'] = pd.to_datetime(optData['Date'], format='%Y-%m-%d')
optData['Maturity'] = pd.to_datetime(optData['Maturity'], format='%Y-%m-%d')
dates = list(optData['Date'].drop_duplicates())
expiries = sorted(list(optData['Maturity'].drop_duplicates()))

filtData = optData[(optData['Date'] == dates[0]) & (optData['Maturity']
== expiries[0])]
S0 = filtData['Underlying'].iloc[0]

print(filtData)

# Run Calibration and generate the plot
def find_cdf(x, Q, B, nu):
    v=1/((1+Q*(np.exp(-B*(x-1))))**(1/nu))
    return 1-v

def find_call_value(S0, K, r, T, p0):
    Q, B, nu = p0
    func = lambda x: find_cdf(x, Q, B, nu)
    p = quad(func, K/S0, np.inf)
    return S0*np.exp(-r*T)*p[0]

def error_func(p0):
    global i, min_RMSE
```

```
    se = []
    for row, option in filtData.iterrows():
        T = (option['Maturity'] - option['Date']).days / 365.
        model_value = find_call_value(S0, option['Strike'], r, T,
p0)
        se.append((model_value - option['Call']) ** 2)
    RMSE = np.sqrt(sum(se) / len(se))
    min_RMSE = min(min_RMSE, RMSE)
    if i % 50 == 0:
        print('%4d |' % i, np.array(p0), '| %7.3f | %7.3f' % (RMSE,
min_RMSE))
    i += 1
    return RMSE

def generate_plot(opt, options):
    options['Model'] = 0.0
    for row, option in options.iterrows():
        T = (option['Maturity'] - option['Date']).days / 365.
        options.at[row, 'Model'] = find_call_value(S0, option['Strike'],
r, T, opt)

    options = options.set_index('Strike')
    fig, ax = plt.subplots(2, sharex=True, figsize=(10, 8))
    options[['Call', 'Model']].plot(style=['b-', 'ro'],
                    title='%s' % str(option['Maturity'])[:10], ax=ax[0])
    ax[0].set_ylabel('option values')
    ax[0].grid(True)
    xv = options.index.values
    se = ((options['Model'] - options['Call'])**2)
    RMSE = np.sqrt(sum(se) / len(se))
    print(RMSE)
    ax[1] = plt.bar(xv - 5 / 2., options['Model'] - options['Call'],
                    width=5)
    plt.ylabel('difference')
    plt.xlim(min(xv) - 10, max(xv) + 10)
    plt.tight_layout()
    plt.grid(True)

i = 0
min_RMSE = 100.
```

```
p0 = sop.brute(error_func, ((6, 10, 0.5), (100, 105, 0.5), (2, 3,
0.5)), finish=None)
print(p0)
opt = sop.fmin(error_func, p0, xtol=0.00001,
                ftol=0.00001, maxiter=750, maxfun=1500)
print(opt)
generate_plot(opt, filtData)

# Plot the extracted PDF and CDF
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

x = np.linspace(0, 3, 1000)
Q, B, nu = opt
cdf = 1/((1+Q*(np.exp(-B*(x-1))))**(1/nu))
pdf = (-1/nu)*((1+Q*np.exp(-B*(x-1)))**((-1/nu)-1))*(-Q*B)*np.exp(-B*(x-1))
plt.rcParams['figure.figsize'] = [10,8]

plt.subplot(211)
plt.plot(x,pdf);
plt.xlabel('S(T)/S(0)');
plt.ylabel('Density');

plt.subplot(212)
plt.plot(x,cdf);
plt.xlabel('S(T)/S(0)');
plt.ylabel('Cumulative Density');

# Calculate the computation time
import time

times = np.zeros(1000)
for i in range(1000):
    t = time.time()
    find_call_value(2905.58, 2850, r, 15/365, opt)
    times[i] = time.time() - t
print(np.mean(times))
```