FINSEARCH REPORT

OPTION PRICING MODELS AND THEIR ACCURACY

GROUP - E - 15

PREPARED BY:-

SHIVAM SONKER - 22B3014

AMISH SETHI - 22B3029

SOMYA GARG - 22B3033

MAHEK HINHORIA-22B3031

TABLE OF CONTENTS

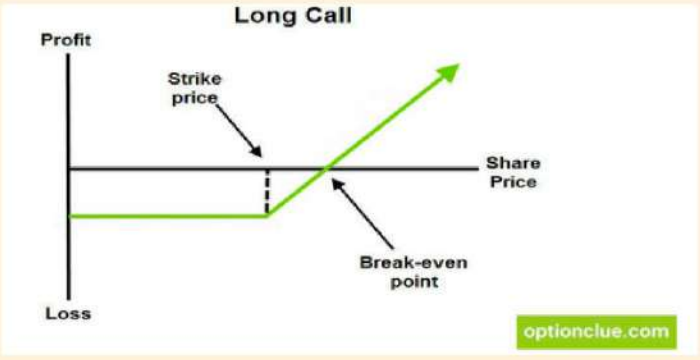
SO, WHAT IS OPTIONS ?

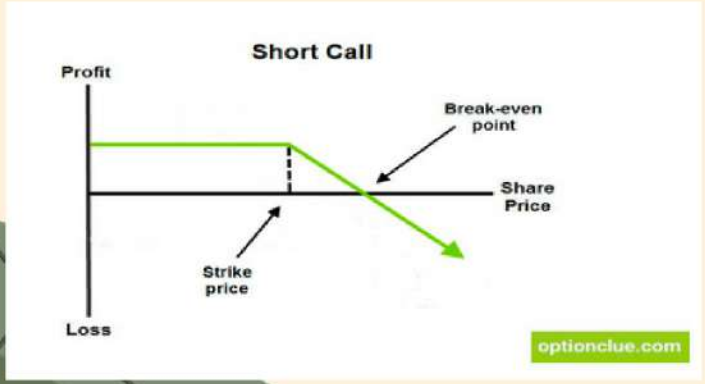
An options contract is a tradable security that grants its owner the right or “option” to buy or sell a predetermined amount of an underlying asset at a specific price on or before a certain date .In exchange of a certain fee called as premium.

CALL OPTION

A type of options contract that gives the holder the right to buy the underlying asset at the strike price before the expiration date.

IV= Spot Price-Strike Price [Spot>=Strike]
- [Spot< strike]

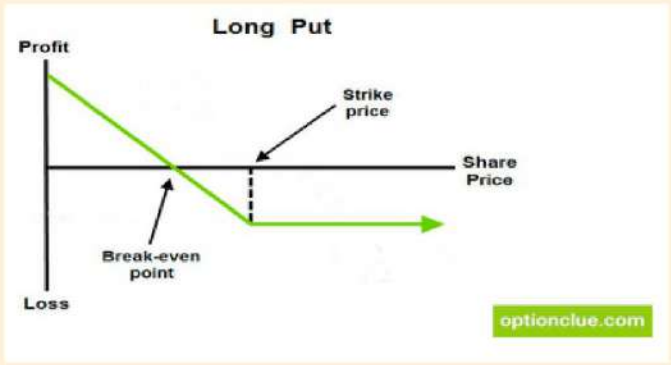


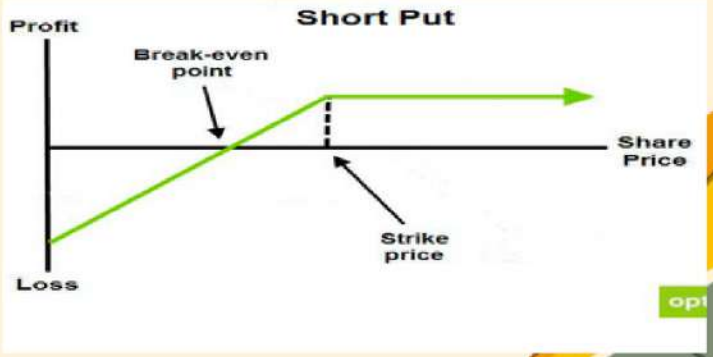


PUT OPTION

A type of option contract that gives the holder the right to sell the underlying asset at the strike price before the expiration date.

IV= Strike Price-Spot Price [Spot =< Strike]
- [Spot > strike]





WHAT IS MONEYNESS OF AN OPTION?

Option moneyness reveals the inherent worth of an option's premium in its present market situation. The moneyness of an option informs you of the potential profitability of your current option contract or the present state of your bet.

MONEYNESS OF OPTIONS

In the Money

Call Option

Current market price of the asset is higher than the option's strike price

This allows the option holder to buy the asset at a lower price than the current market price, making the option valuable.

Put Option

current market price of the asset is lower than the option's strike price

This allows the option holder to sell the asset at a higher price than the current market price making the option valuable.

At the Money

Current price of the underlying asset is exactly equal to the option's strike price.

In this case, the intrinsic value of the option is zero. Both call and put options at the money have the potential to become profitable as the underlying asset's price moves in either direction.

Out of the Money

Call Option

Current market price of the asset is lower than the option's strike price

It would be more expensive to buy the asset at the higher strike price than the current market price, making the option worthless.

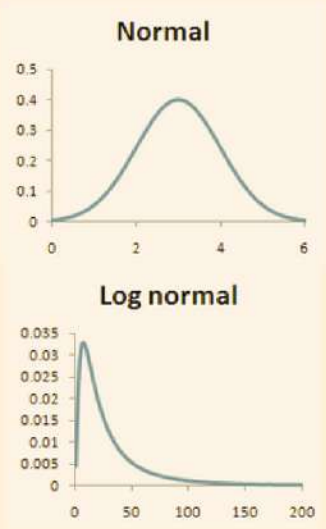
Put Option

Current market price of the asset is higher than the option's strike price

It would be more profitable to sell the asset at the higher market price rather than exercising the option, making the option worthless

BLACK SCHOLES

The Black-Scholes model is one of the most important concepts in modern financial theory. This mathematical equation estimates the theoretical value of derivatives based on other investment instruments, taking into account the impact of time and other risk factors.



Black-Scholes has a lognormal distribution of prices following a random walk with constant drift and volatility. Using this assumption and factoring in other important variables, the equation derives the price of a European-style call option.

The Black-Scholes equation requires five variables :
1)volatility 2) underlying asset 3) the strike price of the option
4) the time until expiration of the option 5) risk-free interest rate.

Underlying Assumptions of the Black-Scholes Model

- **No Dividends**

The Black-Scholes model assumes that the underlying stocks do not pay any dividends or returns.

- **Efficient markets**

The model assumes that financial markets are efficient, meaning that all available information is quickly and accurately reflected in asset prices.

- **Constant risk-free interest rate**

The model assumes that the rate of return of a risk-free investment such as a Treasury bill will remain constant during the life of an option.

- **Constant volatility**

The model assumes that the volatility of the underlying asset's returns remains constant over the option's life.

- **Frictionless market**

The model assumes costless trading, i.e. no transaction costs, including commission, brokerage, and liquidity risks.

- **Expiration date**

The model assumes that the options can only be exercised on its expiration or maturity date. Hence, it does not accurately price American options. It is extensively used in the European options market.

Black-Scholes Equation

$$C = N(d_1)S_t - N(d_2)Ke^{-rt}$$

$$\text{where } d_1 = \frac{\ln \frac{S_t}{K} + (r + \frac{\sigma^2}{2})t}{\sigma\sqrt{t}}$$

$$\text{and } d_2 = d_1 - \sigma\sqrt{t}$$

where:

C=Call option price

S=Current stock (or other underlying) price

K=Strike price

r=Risk-free interest rate

t=Time to maturity

N=A normal distribution

The Volatality Surface

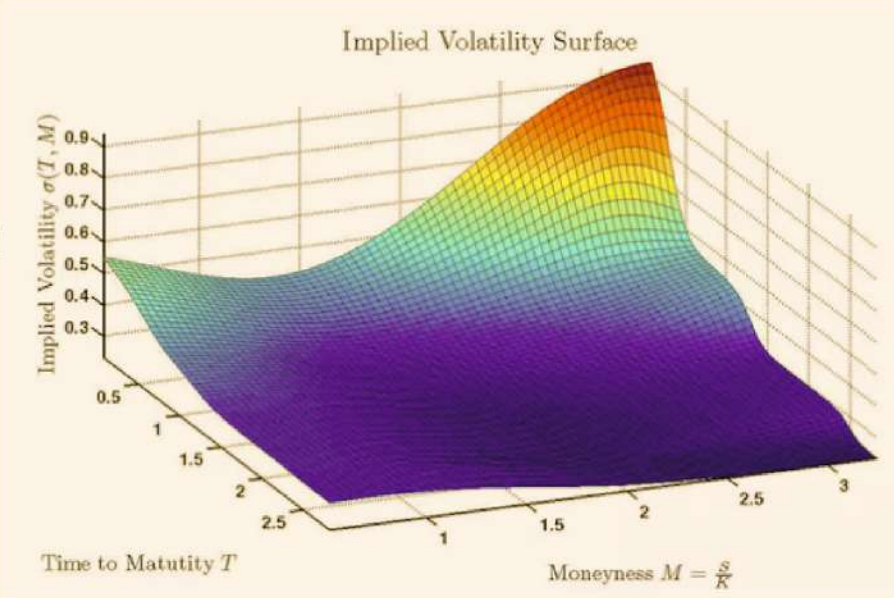
The volatility surface is a three-dimensional plot showing the implied volatilities of a stock's options that are listed on it across different strike prices and expirations.

It is well known that stock prices sometimes go up and down, but this is not always the way the GBM model predicts.If the Black-Scholes model is correct, then we should have a flat implied volatility surface. The volatility surface is a function of price to K and time to maturity T and implicitly defines

$$C(S, K, T) := BS(S, T, r, q, K, \sigma(K, T))$$

where C(S, K, T) - T represents the current market price of a call option with strike price K at expiry and BS (·) is the Black-Scholes price of call option.

The principal features of the volatility surface is that options with lower strikes tend to have higher implied volatilities. For a given maturity, T, this feature is typically referred to as the volatility skew or smile.



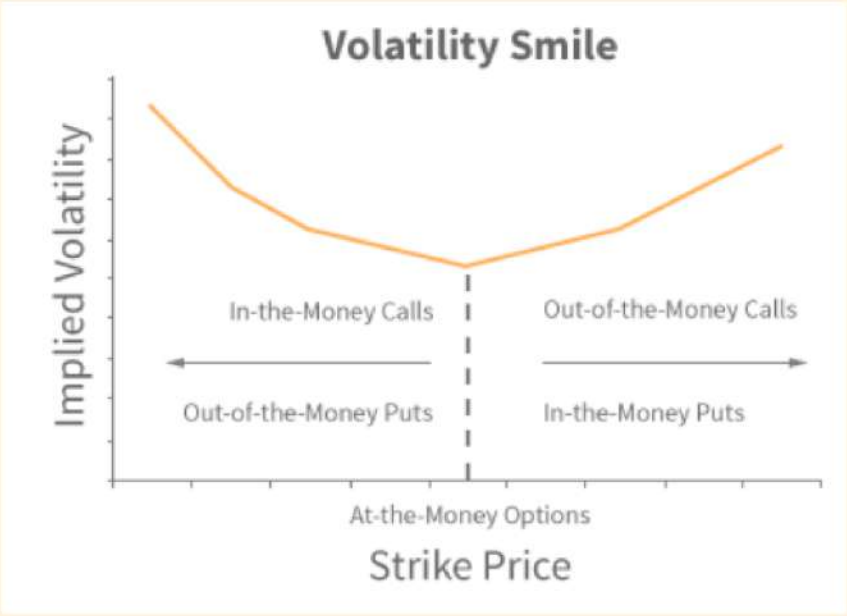
Negative effect for a strike value K can increase or decrease the expiration time. But in general σ(K, T) tends to become a constant as T → ∞. But for small T, we usually look at volatility, where short options are more volatile than long options. This is especially true in times of economic depression.

Why is there a skew or smiley :)

For stocks and stock indices, the shape of the wave surface is always changing. Implied volatility rises when the underlying asset of an option is further out of the money (OTM), or in the money (ITM), compared to at the money (ATM). There are two main explanations for skew.

1. Risk aversion which can appear as an explanation in many guises:
(a) Stocks do not follow GBM with a fixed volatility. Instead they often jump and jumps to the downside tend to be larger and more frequent than jumps to the upside.
(b) As markets go down, fear sets in and volatility goes up.
(c) Supply and demand. Investors like to protect their portfolio by purchasing out-of-the-money puts and so there is more demand for options with lower strikes.

2.The leverage effect which is due to the fact that the total value of company assets, i.e. debt + equity, is a more natural candidate to follow GBM. If so, then equity volatility should increase as the equity value decreases.



Interesting Fact

Before Wall Street crashed in 1987, there was little difference in the market. So it looks like it took twenty years for the business to realize that this was the wrong choice... :)

Put - Call Parity

Let's examine a European call option and a European put option, both featuring identical strike prices denoted as K and sharing the same maturity period represented as T. In the presence of a continuous dividend yield, q, it is noteworthy that put-call parity dictates the following relationship:

$$e^{(-rT)} * K + \text{Call Price} = S + \text{Put Price}.$$

Put-call parity serves as a valuable tool in the computation of Greeks
For example,
it implies that Vega(Call) = Vega(Put) and
that Gamma(Call) = Gamma(Put).
It is also extremely useful for calibrating dividends and constructing the volatility surface.

The Greeks

Financial metrics that traders can use to measure the factors that affect the price of an options contract.

1) Delta

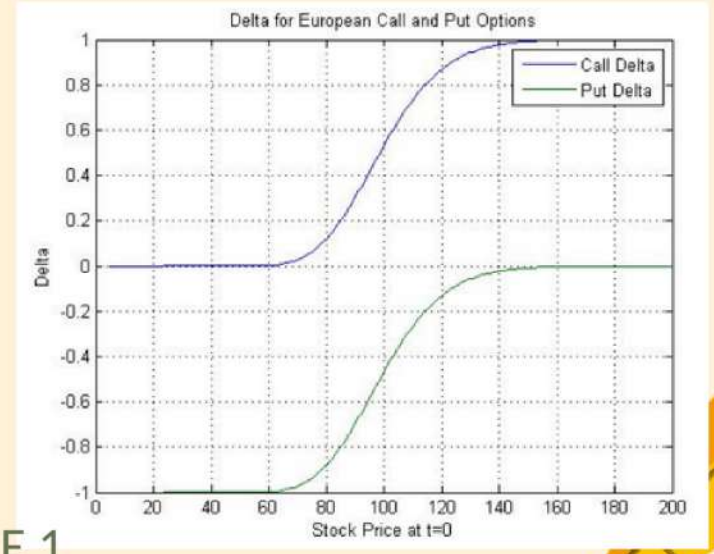
The delta of an option represents the sensitivity to which the price of the option responds to fluctuations in the underlying security's price.

The delta of a European call option satisfies

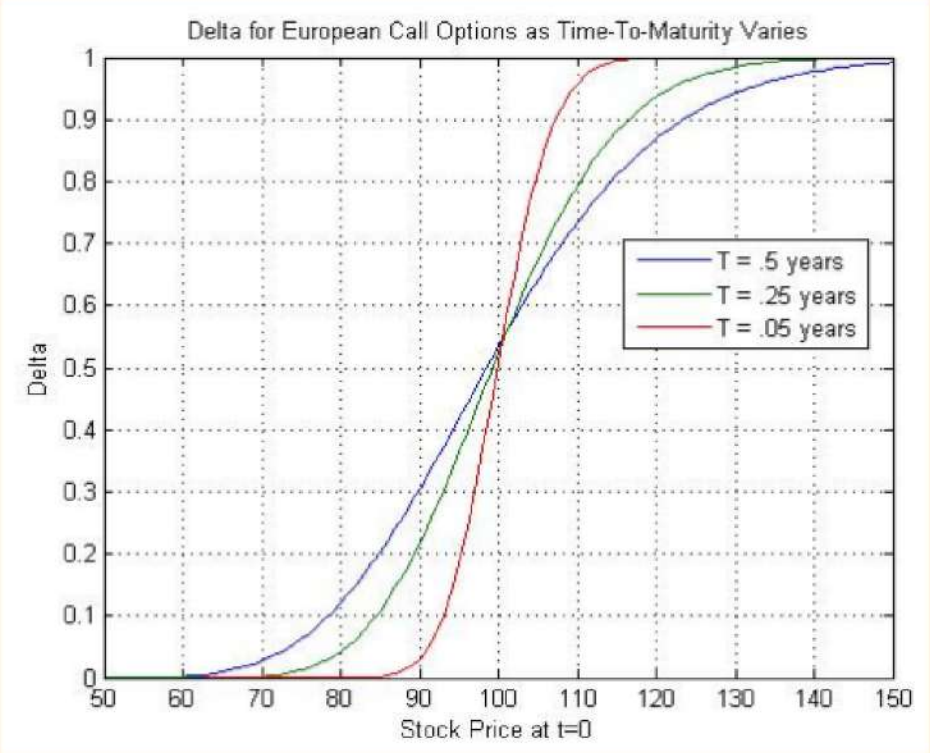
$$\text{delta} = \partial C/\partial S = e^{(-qT)} * \Phi(d1).$$

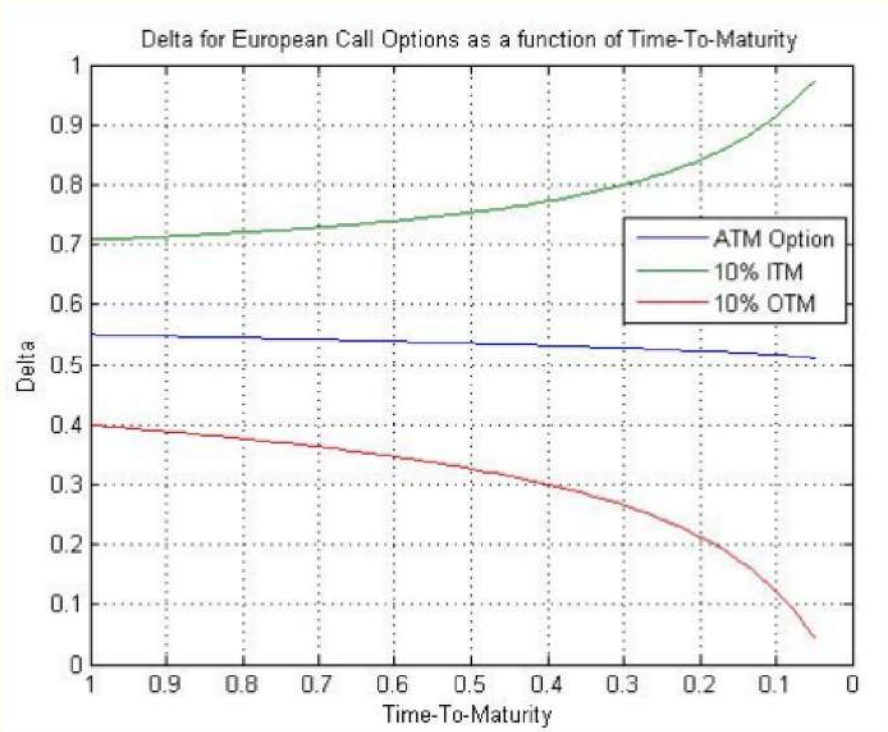
By put-call parity, we have
$\text{delta(put)} = \text{delta(call)} - e^{(-qT)}$

The graph aside illustrates the delta values for both a call option and a put option in relation to the underlying stock price.



The provided graph depicts the delta values for a call option relative to the underlying stock price across three distinct time-to-maturity periods. Notably, the graph highlights that as the time-to-maturity diminishes, the delta exhibits a more pronounced incline in the vicinity of the strike price, K.





Within this graphical representation, we have presented the delta values of a call option, each associated with varying degrees of moneyness, as a function of time-to-maturity.
A better way of understanding this is to consider delta as the probability that the option will expire in-the-money (ITM).

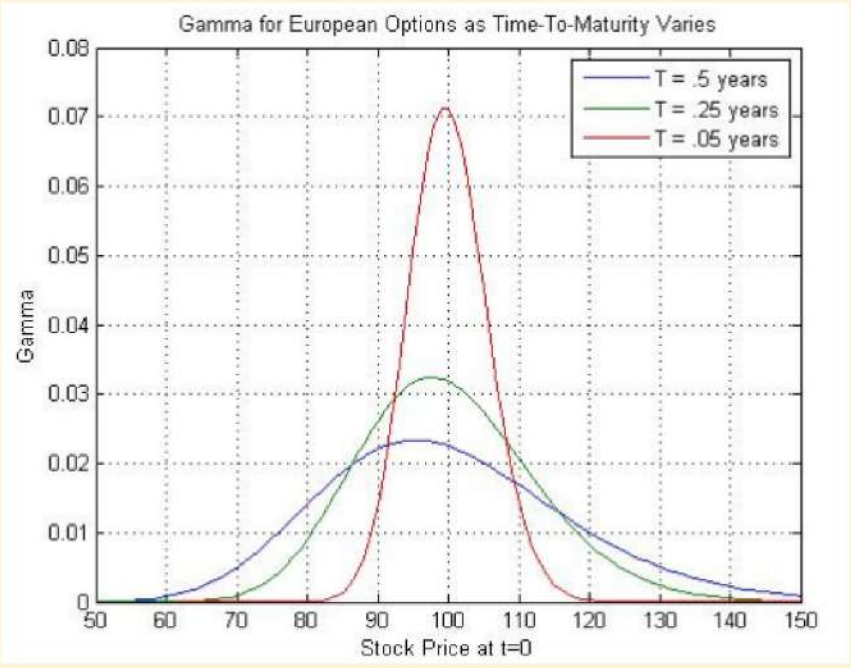
As an option moves beyond its strike price into the in-the-money (ITM) region, the likelihood of it expiring with intrinsic value significantly increases.

2) Gamma

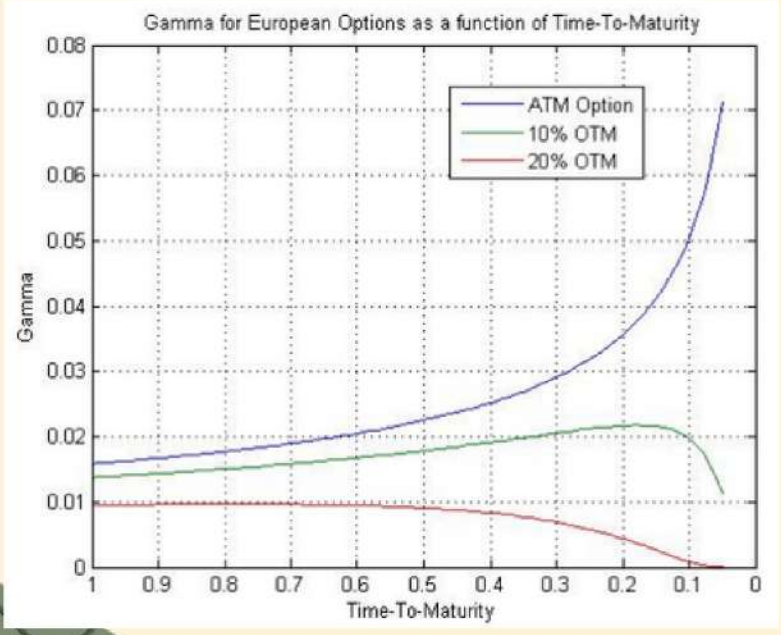
The gamma of an option signifies the responsiveness of the option's delta to alterations in the underlying security's price.

The gamma of a call option satisfies

$$\text{Gamma} = \partial 2/\partial S2(c) = e^{\wedge}(-qT)^{*}\ \varphi(d1)/\ \sigma S\sqrt{T}$$



The chart displays the gamma of a European option relative to the stock price, considering three distinct time-to-maturity scenarios. It's important to observe that, in accordance with put-call parity, the gamma values for European call and put options sharing the same strike price are identical.Gamma is always positive due to option convexity.



Traders who long gamma position have the opportunity to generate profits through a strategy known as gamma scalping. Gamma scalping involves the periodic adjustment of an options portfolio to maintain a delta-neutral position.

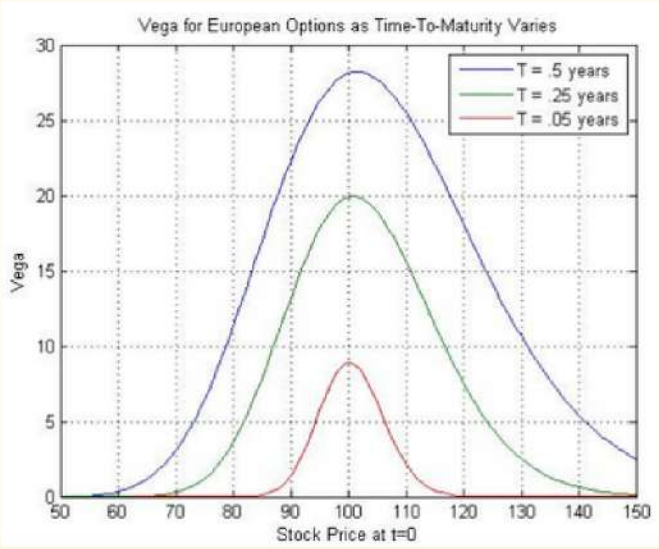
Graph display gamma as a function of time-to-maturity

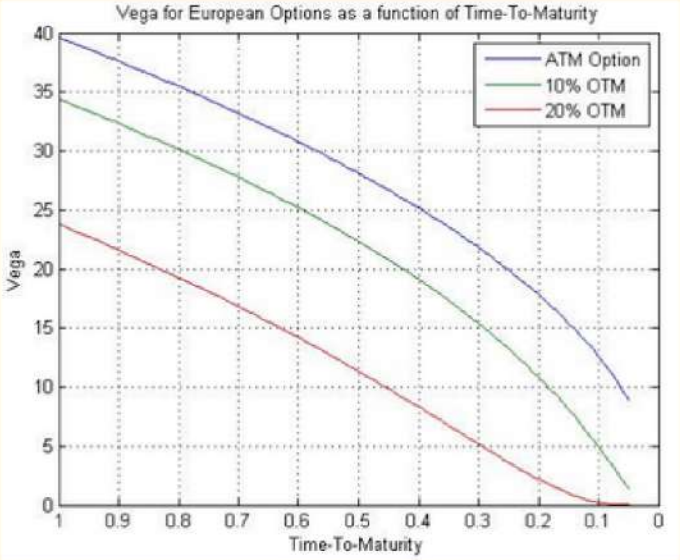
3) Vega

The vega of an option represents the degree to which the option's price responds to fluctuations in volatility.

The vega of a call option satisfies

$$\text{vega} = \partial C/\partial \sigma = e^{\wedge(-qT)} * S \sqrt{T} * \varphi(d1).$$





In plot 2, we observe vega as it relates to the underlying stock price, with the assumption that K = 100 and r = q = 0. It is important to reiterate that, in accordance with put-call parity, the vega of a call option is equivalent to the vega of a put option having the same strike price. Additionally, it's worth noting that vega gradually diminishes to zero as the time-to-maturity approaches zero. This pattern aligns with the observations in Figure 1.

4) Theta

The theta of an option is the sensitivity of the option price to a negative change in time-to-maturity.

LET'S BACKTEST !!!

```
#Extracting required data from NSE
df1=pd.read_csv('Nifty5031Aug.csv')
df1call=df1[['IV','LTP','STRIKE','LTP_put']]
```

Here we extracted data of Nifty50 options expiring on 31st August from NSE website. We took into account the columns - IV(Implied volatility), LTP(Latest traded price),STRIKE(strike price) and LTP_put(last traded price of put options)

```
#Codinig Black-Scholes Model
from scipy.stats import norm
def black_scholes(S, K, T, r, sigma):
    d1 = (np.log(S / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    call_price = S * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)
    return call_price
```

It calculates the theoretical price of a European call option using the Black-Scholes formula and returns this calculated call option price.

```
#Applying the model on a specific Dataset
df1call=df1call.iloc[47:59]
df1call.reset_index(inplace=True)
df1call.drop('index',axis=1,inplace=True)
l=[]
for i in range(0,12):
    a=black_scholes(19400,float((df1call.iloc[i,2]).replace(',','')),7/366,0.0675,float(df1call.iloc[i,0])*0.01)
    l.append(a)
df1call['pred_LTP']=''
df1call['pred_LTP']=l
```

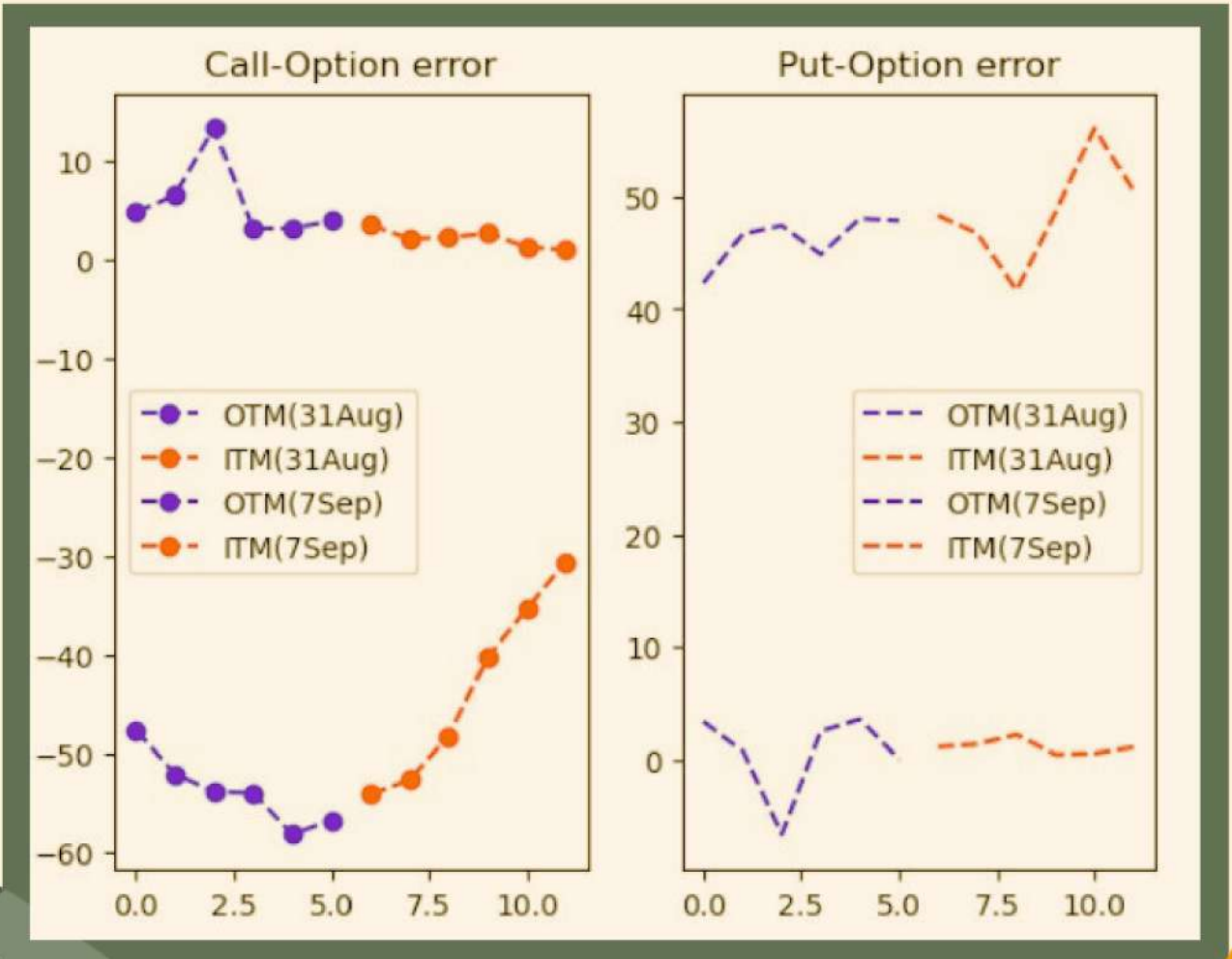
The model is applied iteratively to calculate theoretical call option prices based on the data in these rows, and these calculated prices are stored.

```
#Calculating errors
df1call['error']=''
for i in range(0,12):
    df1call['error'][i]=df1call['pred_LTP'][i]-float(df1call['LTP'][i])
```

An empty 'error' column is created in the DataFrame. Then, a loop iterates through the first 12 rows of the DataFrame, calculating the difference between the predicted call option prices ('pred_LTP') and the actual call option prices ('LTP') for each row.

```
#Plotting the curves
fige,ax=plt.subplots(nrows=1,ncols=2)

ax[0].plot(df1call['error'][0:6],'o--',color='blue')
ax[0].plot(df1call['error'][6:12],'o--',color='red')
```

This code block creates a figure with two subplots and then plots error data from df1call for call options on the first subplot.



```
#Backtesting on 2nd Expiry Date
df2=pd.read_csv('Nifty507SEPT.csv')
df2call=df2[['IV','LTP','STRIKE','LTP_put']]
df2call=df2call.iloc[27:39]
df2call.reset_index(inplace=True)
df2call.drop("index",axis=1,inplace=True)
l2=[]
for i in range(0,12):
    a=black_scholes(19400,float((df2call.iloc[i,2]).replace(',','')),7/366,0.0675,float(df2call.iloc[i,0])*0.01)
    l2.append(a)
df2call['pred_LTP']=''
df2call['pred_LTP']=l2
df2call['error']=''
for i in range(0,12):
    df2call['error'][i]=df2call['pred_LTP'][i]-float(df2call['LTP'][i])
ax[0].plot(df2call['error'][0:6],'o--',color='blue')
ax[0].plot(df2call['error'][6:12],'o--',color='red')
plt.title('Error plot')
ax[0].set_title("Call-Option error")
ax[0].legend(['OTM(31Aug)','ITM(31Aug)','OTM(7Sep)','ITM(7Sep)'])
```

```
#Using Put Call Parity for calculating Price of Put Options

df1call['put_p']=''

arr=[]
for i in range(0,12):
    a=float((df1call['STRIKE'][i]).replace(',',''))
    arr.append(a)
for i in range(0,12):
    df1call['put_p'][i]=float(df1call['LTP'][i])+arr[i]*(pow(m.e,-0.0675*7/366))-19368.


df1call['error_put']=df1call['put_p']-(df1call['LTP_put']).astype(float)
print(df1call)


df2call['put_p']=''

arr2=[]
for i in range(0,12):
    a=float((df2call['STRIKE'][i]).replace(',',''))
    arr2.append(a)
for i in range(0,12):
    df2call['put_p'][i]=float(df2call['LTP'][i])+arr2[i]*(pow(m.e,-0.0675*7/366))-19368.


df2call['error_put']=df2call['put_p']-(df2call['LTP_put']).astype(float)
print(df2call)
```

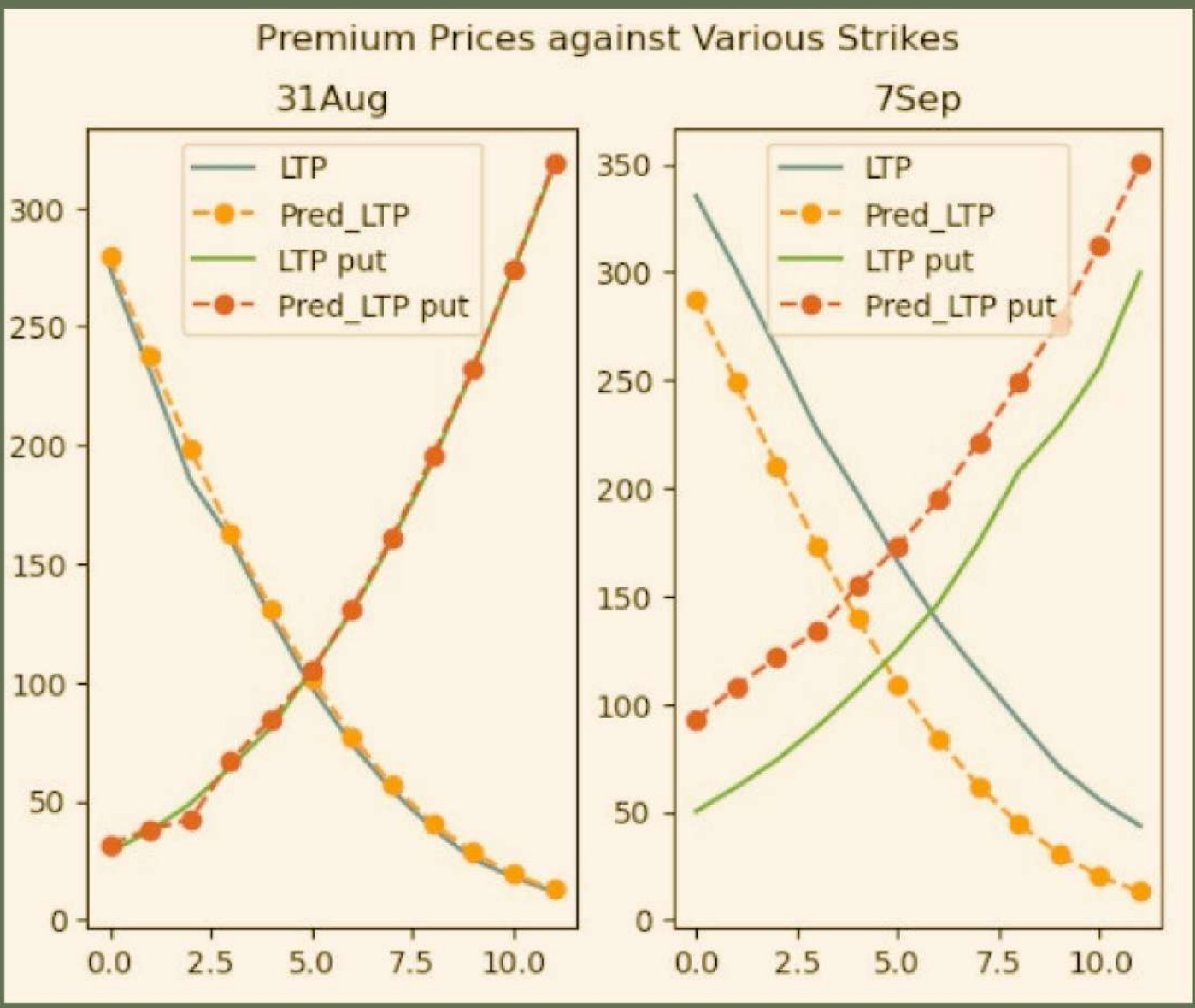
This code block calculates put option prices using the Put-Call Parity relationship for two datasets, df1call and df2call. It derives the put option prices ('put_p') by adjusting the call option prices ('LTP') and strike prices ('STRIKE') based on the Parity formula.

```
#Plotting errors for put options
ax[1].plot(df1call['error_put'][0:6],'b--')
ax[1].plot(df1call['error_put'][6:12],'r--')
ax[1].plot(df2call['error_put'][0:6],'b--')
ax[1].plot(df2call['error_put'][6:12],'r--')
ax[1].legend(['OTM(31Aug)','ITM(31Aug)','OTM(7Sep)','ITM(7Sep)'])
ax[1].set_title("Put-Option error")

plt.show()
fige.suptitle("Deviation from actual premiums against Various Strikes")
```

This code block plots the errors in put option price calculations. It displays error data for out-of-the-money (OTM) and in-the-money (ITM) put options on the same plot, with different colors and line styles.

```
#Plotting predicted LTP's and actual LTP's
fig,axes = plt.subplots(nrows=1,ncols=2)

axes[0].plot(df1call['LTP'].astype(float))
axes[0].plot(df1call['pred_LTP'],'o--')
axes[0].set_title('31Aug')

axes[0].plot(df1call['LTP_put'].astype(float))
axes[0].plot(df1call['put_p'],'o--')
axes[0].legend(['LTP','Pred_LTP','LTP put','Pred_LTP put'])
axes[0].set_title('31Aug')

axes[1].plot(df2call['LTP'].astype(float))
axes[1].plot(df2call['pred_LTP'],'o--')
axes[1].set_title('7Sept')

axes[1].plot(df2call['LTP_put'].astype(float))
axes[1].plot(df2call['put_p'],'o--')
axes[1].legend(['LTP','Pred_LTP','LTP put','Pred_LTP put'])
axes[1].set_title('7Sep')

fig.suptitle("Premium Prices against Various Strikes")
```

This code block creates a figure with two subplots. In the first subplot (axes[0]), it plots actual and predicted call and put option prices for a specific date ('31Aug'). In the second subplot (axes[1]), it does the same for a different date ('7Sept'). The legend labels the lines, and the super-title describes the visualization as "Premium Prices against Various Strikes."




```
# Calculating RMSE Errors for the results and creating an RMSE dataframe
from sklearn.metrics import mean_squared_error
R=m.sqrt(mean_squared_error(df1call['pred_LTP'],df1call['LTP'].astype(float)))
b=m.sqrt(mean_squared_error(df2call['pred_LTP'],df2call['LTP'].astype(float)))
c=m.sqrt(mean_squared_error(df1call['put_p'],df1call['LTP_put'].astype(float)))
d=m.sqrt(mean_squared_error(df2call['put_p'],df2call['LTP_put'].astype(float)))

data={'31Aug':[R,c],'7Sep':[b,d]}
RMSE=pd.DataFrame(data,index=['call','put'])

print(RMSE)
```

This code block calculates the Root Mean Square Error (RMSE) for the model's predictions of call and put option prices. It uses the mean_squared_error function from scikit-learn.

	31Aug	7Sep
call	5.010757	49.308931
put	2.653916	47.544610

RMSE Matrix

	IV	LTP	STRIKE	...	error	put_p	error_put
0	6.64	274.55	19,150.00	...	4.744741	31.843615	3.343615
1	7.42	231	19,200.00	...	6.420944	38.229108	0.729108
2	7.82	185.15	19,250.00	...	13.191121	42.3146	-6.6854
3	8.12	159.95	19,300.00	...	3.078237	67.050093	2.500093
4	8.26	127.65	19,350.00	...	3.090696	84.685585	3.535585
5	8.3	98	19,400.00	...	3.855492	104.971077	-0.028923
6	8.34	74	19,450.00	...	3.421648	130.90657	1.10657
7	8.3	54.5	19,500.00	...	2.049039	161.342062	1.342062
8	8.35	38.55	19,550.00	...	2.195259	195.327555	2.177555
9	8.36	25.7	19,600.00	...	2.57355	232.413047	0.413047
10	8.39	17.95	19,650.00	...	1.191609	274.59854	0.49854
11	8.46	11.85	19,700.00	...	0.940315	318.434032	1.084032

31st AUG Final dataset

	IV	LTP	STRIKE	...	error	put_p	error_put
0	8.64	335.35	19,150.00	...	-47.55936	92.643615	42.293615
1	9.24	301	19,200.00	...	-52.015698	108.229108	46.679108
2	9.35	264.3	19,250.00	...	-53.792158	121.4646	47.4146
3	9.21	227	19,300.00	...	-53.920387	134.100093	44.850093
4	9.12	197.6	19,350.00	...	-58.138022	154.635585	48.035585
5	9	166	19,400.00	...	-56.700667	172.971077	47.871077
6	8.95	138	19,450.00	...	-54.082785	194.90657	48.30657
7	8.84	114.65	19,500.00	...	-52.60806	221.492062	46.742062
8	8.74	92.55	19,550.00	...	-48.190431	249.327555	41.627555
9	8.65	70.8	19,600.00	...	-40.194802	277.513047	48.613047
10	8.55	55.55	19,650.00	...	-35.34587	312.19854	56.04854
11	8.51	43.6	19,700.00	...	-30.546643	350.184032	50.434032

7th SEPT Final dataset

Hedging an option through BSM

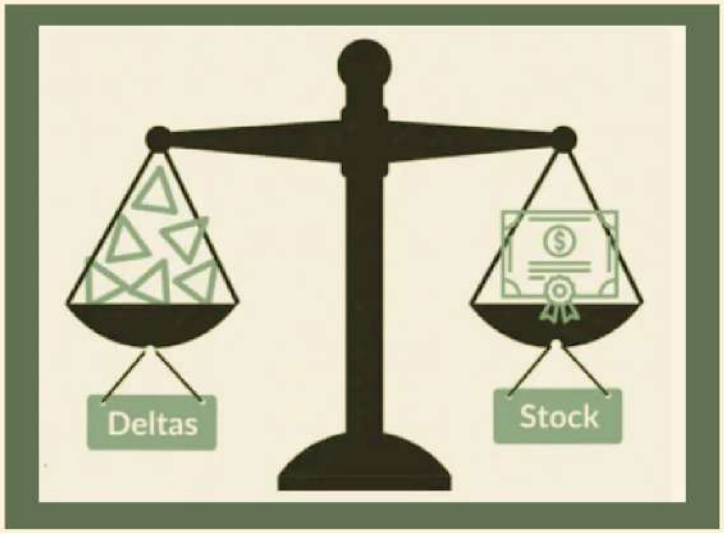
The Black-Scholes formula can be used to create a hedge for an option if the holder of the option wants to limit their exposure to the variations in the index. This can be done via a strategy called delta hedging, which i simply means taking a certain short position on the index.

In this manner, the movements in the call price will be compensated by the movements in the short position on the index, regardless of the direction of the changes in the underlying asset price.

This strategy reduces the investors' return if the index goes up but reduces their loss if it goes down as well. In other words, the strategy reduces the investors' risk at the cost of less expected return.

A complete hedge (for both effects) for a call option can be achieved by using a portfolio invested in the underlying asset and the risk-free asset only. It is useful to realise that, at the end of the day, finding a perfect hedge for the option is equivalent to finding a replicating portfolio for the option, on which the investor can take a short position to achieve their hedge.

From the Black-Scholes formula, the theoretical replicating portfolio P for a call option can be achieved by buying Δ shares in the underlying asset S and taking a short position b on the riskless asset B:



$$P = \triangle S - bB$$

$$b = 1/2[\Theta + 1/2(\sigma^2 * S^2{}_T)]$$

Where S is the underlying price, sigma is the implied volatility, r is the risk-free rate and delta, gamma and theta are three of the greeks (partial derivatives of the call price C)

$$\Delta = \frac{\partial C}{\partial S} \qquad \Gamma = \frac{\partial^2 C}{\partial S^2} \qquad \Theta = \frac{\partial C}{\partial t}$$

The weights change over time. The more frequently the weights are updated, the more accurate the replicating portfolio will be.

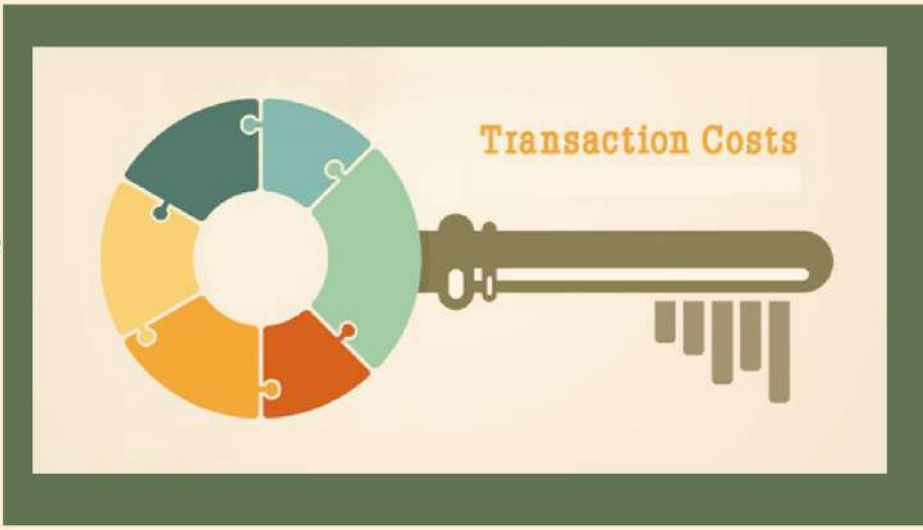
In order to evaluate the accuracy of this method in discrete time, a replicating portfolio for the considered call is constructed in this way. The portfolio weights are updated only once every trading day, using closing prices. The returns on this portfolio are calculated from these weights and the returns on the index and the riskless asset:S&P 500 index

As you can see, the replicating portfolio is a reasonable approximation for the call option but the errors can be significant after a certain amount of time.

As a conclusion , we can say even under the easiest case assumptions that implied volatility remains constant during the life of the option and that a perfect riskless asset exists, the Black-Scholes theoretical hedge may be quite inaccurate in discrete time.

A more frequent update in the portfolio weights together with more advanced hedging strategies are used for a more precise hedge on an option. These strategies cover against changes in implied volatility and include second-order approximations (see vega and gamma hedging). Such hedging strategies may also use puts and other derivatives.

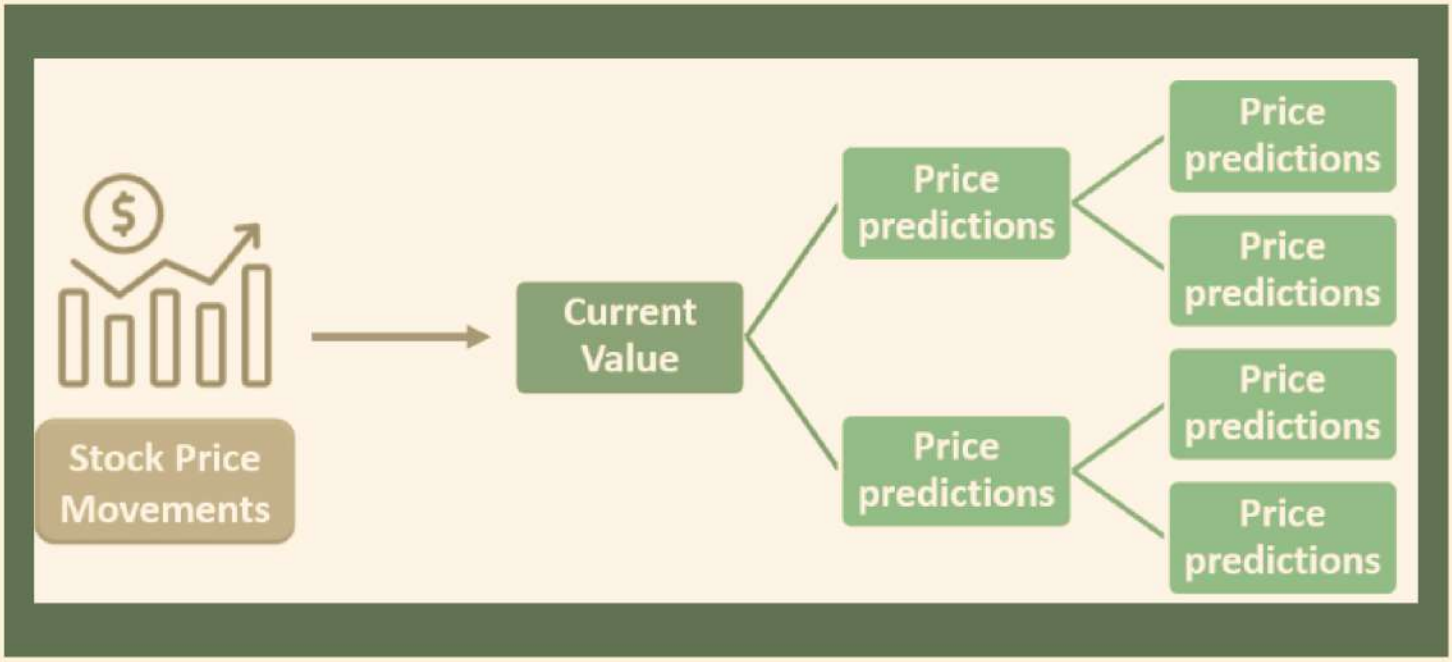
Transaction costs and options premiums should also be considered as they can spoil the accuracy of a hedge when considering the net return on the portfolio. Transaction costs create an interesting trade-off between the accuracy of a hedging strategy and its costs.



BINOMIAL MODEL ?

Binomial option pricing model is a very simple model that is used to price options.This model is based on the concept of no arbitrage.

The binomial option pricing model is a risk-free method for estimating the value of path-dependent alternatives. With this model, investors can determine how likely they are to buy or sell at a given price in the future. According to this model, the current option value is equal to the present value of the probability-weighted future payoffs of the investment.



From an Investor's POV

An investor is aware of the current stock price at any given time. They're going to try to predict future changes in stock prices. They will divide the time until the option expires into equal parts under this scenario (weeks, months, quarters).

The model uses an iterative process for each period to determine how likely the movement will be up or down. The model effectively creates a binomial distribution of stock prices.

Binomial Tree

The best approach to visualize the model is using a binomial tree.

```
# Initialise parameters
S0 = 100      # initial stock price
K = 100       # strike price
T = 1         # time to maturity in years
r = 0.06      # annual risk-free rate
N = 3         # number of time steps
u = 1.1       # up-factor in binomial models
d = 1/u       # ensure recombining tree
opttype = 'C' # Option Type 'C' or 'P'
```

Binomial Tree Representation

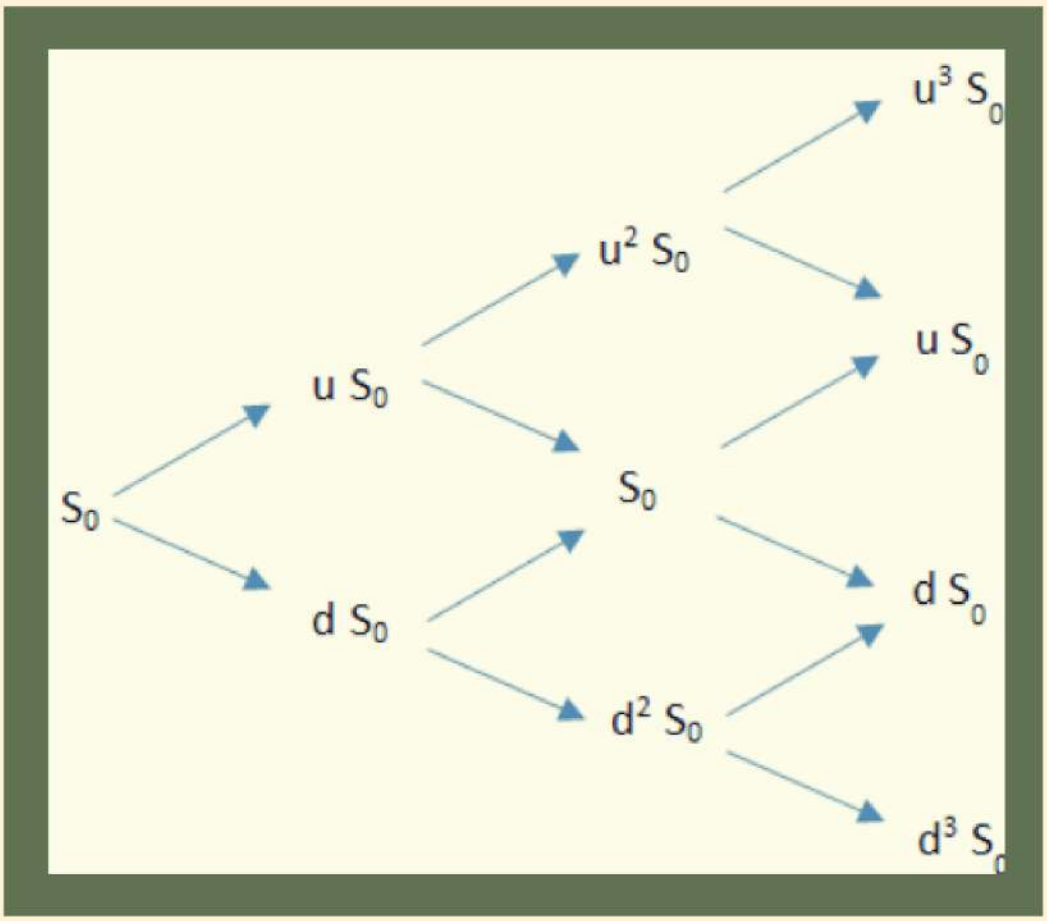
Stock tree can be represented using nodes (i,j) and intial stock price so-

$S(i,j)=S(0)u^{(j)}d^{(i-j)}$

C(i,j) represents contract price at each node (i,j). Where C(N,j) represents final payoff function that we can define.

For this we will price a European Call, so-

$$C(N,j)=\max(S(N,j)-K,0)$$



Example

Q) A stock is worth 60 Rs today. In a year the stock price can rise or fall by 15 percent. The interest rate is 6%. A put option expires in one years and has an exercise price of 60 Rs.

The formula for π (pi) is :

$\pi = (1 + r - d) / (\mu - d)$

The risk-neutral probability is $\pi = (1.06 - 0.85) / (1.15 - 0.85) = 0.7$, and $1 - \pi = 0.3$.

Stock prices in the binomial tree one years from now are:

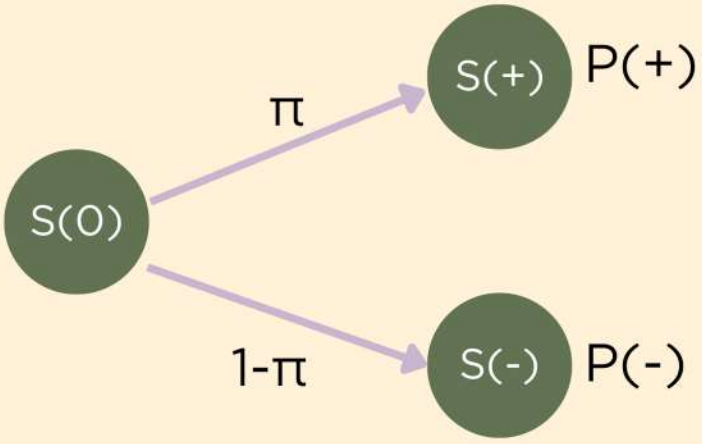
- S+ = 60 (1.15) = 69 Rs.
- S- = 60 (0.85) = 51 Rs.

The option prices at the end of year 1:

p+ = (0.7 x 0 + 0.3 x 1.35)/(1.06) = 0.3821 Rs.

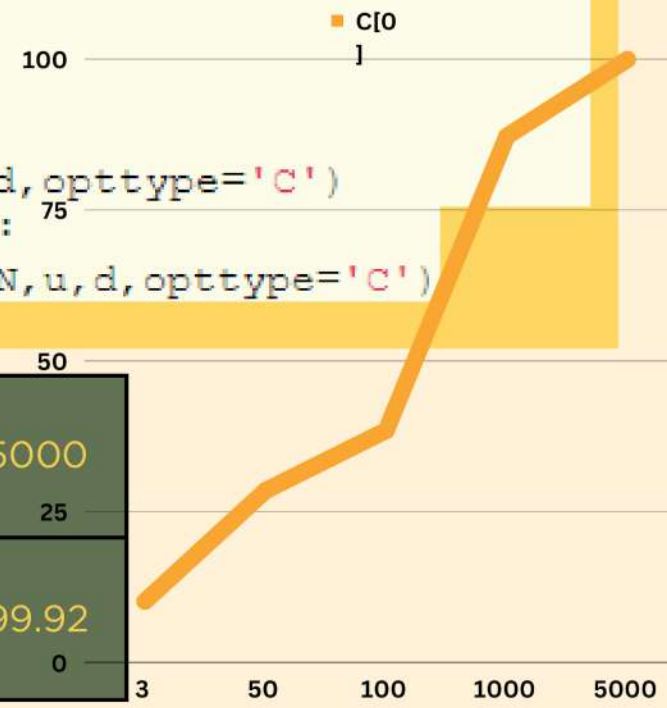
p- = (0.7 x 1.35 + 0.3 x 16.65)/(1.06) = 5.60 Rs.

The put price today is p = (0.7 x 0.3821 + 0.3 x 5.6)/1.06 = 1.83 Rs



Prices at different Time Frames

```
def binomial_tree_slow(K,T,S0,r,N,u,d,opttype='C'):
    #precompute constants
    dt = T/N
    q = (np.exp(r*dt) - d) / (u-d)
    disc = np.exp(-r*dt)

    # initialise asset prices at maturity - Time step N
    S = np.zeros(N+1)
    S[0] = S0*d**N
    for j in range(1,N+1):
        S[j] = S[j-1]*u/d

    # initialise option values at maturity
    C = np.zeros(N+1)
    for j in range(0,N+1):
        C[j] = max(0, S[j]-K)

    # step backwards through tree
    for i in np.arange(N,0,-1):
        for j in range(0,i):
            C[j] = disc * ( q*C[j+1] + (1-q)*C[j] )

    return C[0]

binomial_tree_slow(K,T,S0,r,N,u,d,opttype='C')
 for N in [3,50, 100, 1000, 5000]:
     binomial_tree_slow(K,T,S0,r,N,u,d,opttype='C')
```

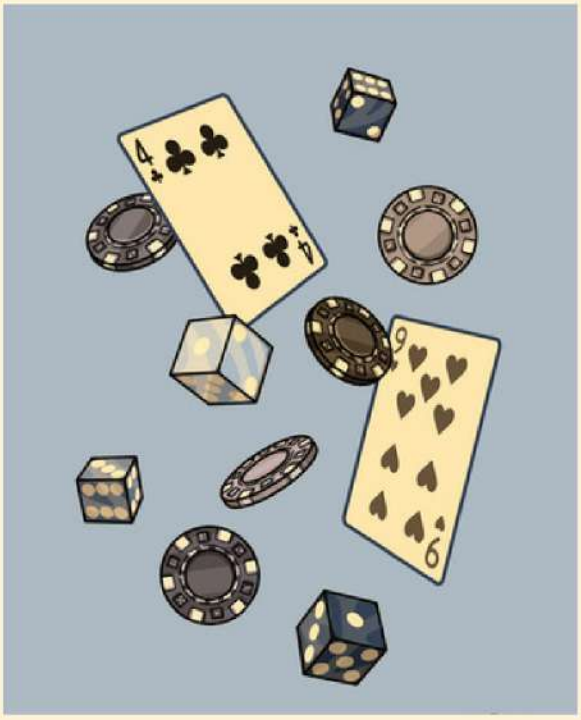


Time (days)	3	50	100	1000	5000
C[0]	10.14	28.49	38.45	87.19	99.92

MONTE CARLO SIMULATION

Monte Carlo simulation is a computing approach that uses random sampling and statistical analysis to estimate the outcomes of complicated systems or situations. It is named after the famed Monte Carlo Casino in Monaco, known for games of chance that rely on unpredictability.

The technique of Monte Carlo simulation entails creating a large number of random samples or scenarios that reflect the possible values of unknown variables in a system. These samples are then utilised to run simulations or computations to examine the system's behaviour and consequences.



Application in finance

Due to the complexity and uncertainty of financial markets and products, Monte Carlo simulation is extremely significant in finance. It offers useful insights and assists financial professionals in making educated decisions, managing risks, and estimating the prospective outcomes of various financial scenarios. Here are some of the most important applications of Monte Carlo simulation in finance:

Options and other derivatives pricing models
Company valuation (to cross-check results obtained through a DCF, use Monte Carlo to triangulate results)
Risk management (Monte Carlo is frequently used for Value at Risk estimation and volatility modelling)
Insurance Pricing: Monte Carlo simulation is used in the insurance sector to estimate policy premiums and reserves while accounting for unpredictable occurrences such as accidents, natural catastrophes, and health risks

The large number of repetitions involved in a Monte Carlo simulation is significantly facilitated by the strong advents of computing power we have observed in recent years.

This technique is instrumental whenever an analyst would like to understand the different possible realizations that could be obtained through a given distribution function. For example, if a distribution function considers three possible variables influencing an asset's price, then by testing for different values of these variables, we could obtain a good idea about the possible prices of the asset (provided that the distribution function represents well , the development of the asset's price).

Option Pricing -

The basic idea used for option pricing is to generate random future scenarios of the underlying asset's price and use these scenarios to estimate the option's value.

Process Involved in Option Pricing -

Stock Price Prediction

We utilise Monte Carlo simulation to simulate how stock prices could vary over time. The concept is easy to understand. First, we determine the stock's return pattern and divide it into discrete time intervals. The potential price changes for the stock are then determined using these processes.

As an illustration, stock returns frequently exhibit a geometric Brownian motion. In order to represent this, we utilise a particular equation (Equation 1) together with a Wiener process (Wt). By applying Itô's formula, we get another equation (Equation 2) that helps us predict stock prices using Monte Carlo simulation, with a variable (Zt) following a standard normal distribution.

$$dS_t = \mu S_t dt + \sigma S_t dW_t \quad \text{..... (1)}$$

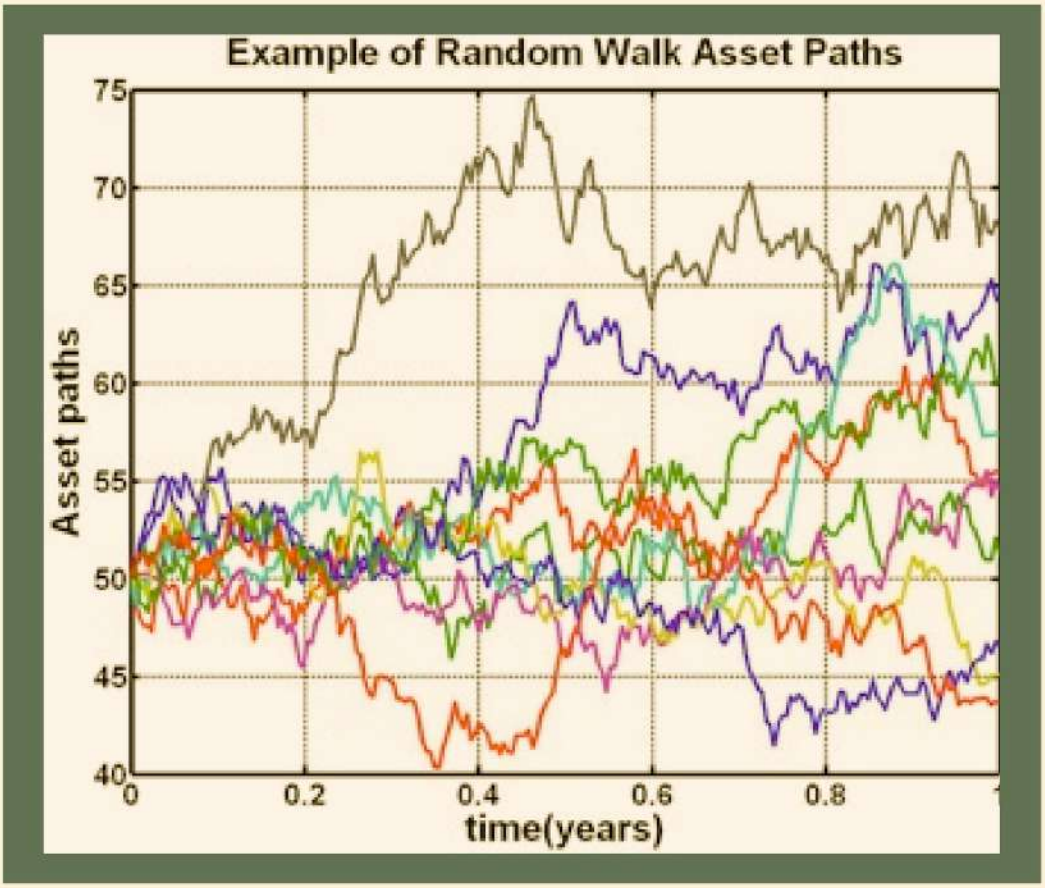
$$S(\Delta t) = S(0)\exp\left|\left(\mu - \frac{\sigma^2}{2}\right)\Delta t + \left(\sigma\sqrt{\Delta t}\right)\varepsilon\right|$$

where,

- S(0): The stock price today.
- S(Δt): The stock price at a (small) time into the future.
- Δt: A small increment of time.
- μ: The expected return.
- σ: The expected volatility.
- ε: A (random) number sampled from a standard normal distribution.

The basic idea used for option pricing is to generate random future scenarios of the underlying asset's price and use these scenarios to estimate the option's value.

Equation 2 may be used repeatedly to produce a variety of probable asset pathways (between now and expiration). The graph below illustrates 10 such pathways as an example. Equation 2 is used to calculate the underlying price at each time step along each path by repeatedly sampling from a typical normal distribution.



Calculation of Option Payoff:

Once the asset paths have been simulated they are used to price the option according to the option's payoff formulae.For each generated price path, calculate the option's payoff at the expiration date using the formulae-->

$$\text{Payoff}_{\text{call}} = \max(A - X, 0)$$

$$\text{Payoff}_{\text{put}} = \max(X - A, 0)$$

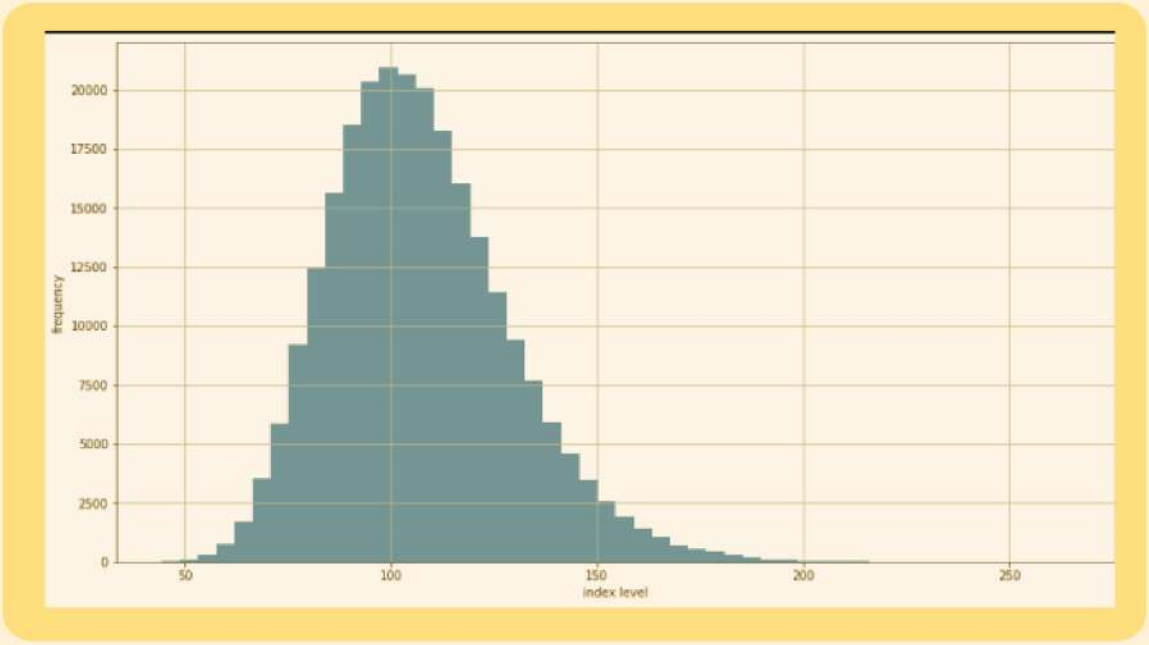
where A is the average value of the asset price over the life of the option and X is the strike.

Discount future payoffs: Since the payoff is at expiration, it needs to be discounted back to the present value using the risk-free rate. The present value of the option is the average of all discounted payoffs. The more simulations you run, the more accurate the pricing estimate becomes.

Monte Carlo Simulation (Python)

Our next look off is on how we simulated the Monte Carlo method on python . As we already know the equation , we tried to implement it in our python program , by taking initial price as S0 and divding the total time into 50 steps and creating 250000 total possible paths for the random walk of the index.

```
import math
from numpy import *
from time import time
# star import for shorter code
random.seed(20000)
t0 = time()
# Parameters
S0 = 100.; SK = 105.; T = 1.0; r = 0.067; sigma = 0.2
M = 50; dt = T / M; I = 250000
# Simulating I paths with M time steps
S = S0 * exp(cumsum((r - 0.5 * sigma ** 2) * dt
+ sigma * math.sqrt(dt)
* random.standard_normal((M + 1, I)), axis=0))

# if only the final values are of interest
S[0] = S0
# Calculating the Monte Carlo estimator
# C0 is the dicounted gain from the option
C0 = math.exp(-r * T) * sum(maximum(S[-1] - SK, 0)) / I
# Results output
tnp2 = time() - t0

print('The payoff is ', C0)
print('The Execution Time is: ',tnp2)

import matplotlib.pyplot as plt
plt.plot(S[:, :20])
plt.grid(True)
plt.xlabel('Steps')
plt.ylabel('Index level')
plt.show()

plt.rcParams["figure.figsize"] = (15,8)
plt.hist(S[-1], bins=50)
plt.grid(True)
plt.xlabel('index level')
plt.ylabel('frequency')
```



This graph consists of the distribution of final index level after completion of all steps.

Backtesting the Sample Code -

We tried to back test our simulation on the historical data of Nifty 50 and we got the following curves

Code for Backtesting

```
import math
import numpy as np
import pandas as pd
from numpy import *
from time import time
import matplotlib.pyplot as plt
# star import for shorter code
random.seed(20000)
t0 = time()
# Parameters
S0 = 18297. #Index Price at step 0
T = 1.0/50;
r = 0.067 #risk free return rate
sigma = 0.11 #volatility
M = 50; dt = T / M; I = 250000
step=M
L=[] #empty list for storing averaged values after each step
# Simulating I paths with M time steps
for i in range(0,step):

  S = S0 * exp(cumsum((r - 0.5 * sigma ** 2) * dt
  + sigma * math.sqrt(dt)
  * random.standard_normal((M + 1, I)), axis=0))
  S[0]=S0

  L.append(S[-1].mean())
  S0=S[-1].mean()  #Updating index at each step

print(S0) #Final predicted value of index

#Importing historic data for backtesting
df=pd.read_csv("^NSEI.csv")
df=df.tail(50)
df.reset_index(drop=True,inplace=True)

#Plotting the historic data and prediced path on same graph
import matplotlib.pyplot as plt
plt.plot(L,label="Predicted path",color='blue')
plt.plot(df['Close'], label="Actual Path",color='green')
plt.grid(True)
plt.xlabel('Days')
plt.ylabel('Nifty 50')
plt.legend()
plt.show()

#calculating RMSE for the data
MSE = np.square(np.subtract(L,df['Close'])).mean()
RMSE = math.sqrt(MSE)
print("Root Mean Square Error:")
print(RMSE)
```

Back Testing Results

```
19591.94467139504
Root Mean Square Error:
205.1144532673078
```

In the code above we adjusted the volatility to 0.11 according to Nifty VIX and current risk free return rate in India to 0.067 . Also we divided each step into further 50 steps then calculated the end value of each step and calculated mean for all I paths. Thus repeating the above for all steps and plotting it we got the predicted curve.

Root Mean Square Accuracy

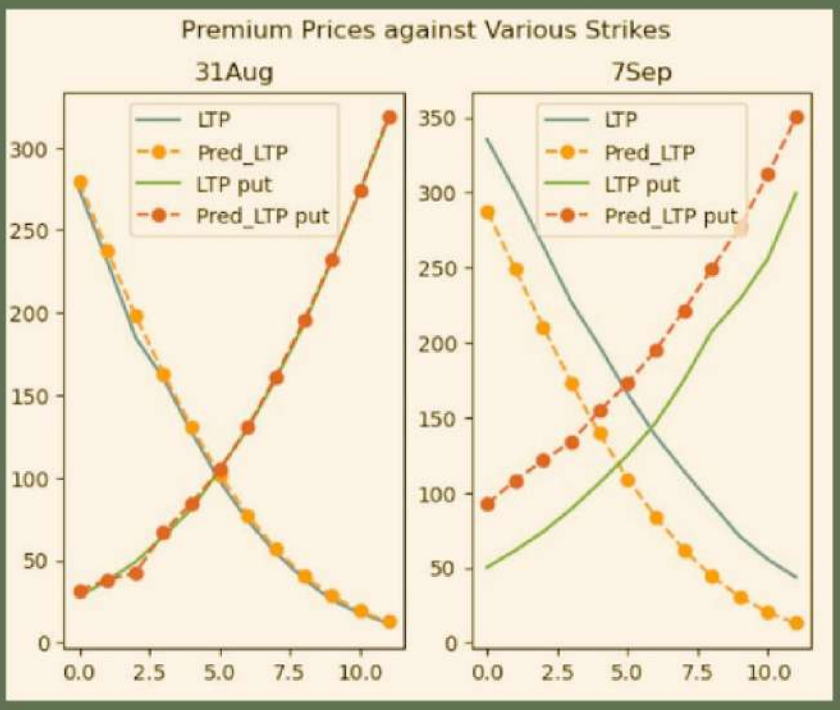
$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

RMSE

So, as seen above, Root Mean Square Error is the square root of the average of the squared differences between the estimated and the actual value of the variable/feature.

RMSE value is calculated to know the efficiency of the model . RMSE values around 200 are considered to be good/moderate .

CONCLUSION



Call and Put Errors

error	error_put
4.744741	3.343615
6.420944	0.729108
13.191121	-6.6854
3.078237	2.500093
3.090696	3.535585
3.855492	-0.028923
3.421648	1.10657
2.049039	1.342062
2.195259	2.177555
2.57355	0.413047
1.191609	0.49854
0.940315	1.084032

RMSE Matrix

	31Aug	7Sep
call	5.010757	49.308931
put	2.653916	47.544610

Option pricing models are essential tools which enables investors to assess the fair value of options and make informed investment decisions Though the Models have some limitation and there were errors in each of them , they were able to give us a fair idea.

We have illustrated Black-Scholes ,binomial and monte-carlo model, which value options by creating replicating portfolios composed of the underlying asset and riskless lending or borrowing. These models can be used to value assets that have option-like characteristics.

The error increases as the time period increases.

REFERENCES

- http://www.columbia.edu/~mh2078/FoundationsFE/BlackScholes.pdf
- https://www.investopedia.com/
- https://zerodha.com/varsity/module/option-theory/
- https://corporatefinanceinstitute.com/topic/derivatives/
- https://www.investopedia.com/terms/m/montecarlosimulation.asp
- https://www.tejwin.com/en/insight/options-pricing-with-monte-carlo-simulation/
- https://www.investopedia.com/terms/b/blackscholes.asp
- https://www.wallstreetmojo.com/binomial-option-pricing-model/
- https://efinancemanagement.com/derivatives/option-pricing-model
- https://quantpy.com.au/binomial-tree-model/intro-to-binomial-trees/
- https://www.simplilearn.com/binomial-option-pricing-model

OUR TEAM

Shivam Sonker

Mahek Hinhoriya

Amish Sethi

Somya Garg

THANK YOU !

For giving us this wonderful opportunity. We really learned a lot ....❤