Hunting for smiles

Francesco Baiano Leonardo Baldassarri Mattia Chiappari
Andrea Maccarrone Hugo Paolini Gianluca Regoli

Applied Numerical Finance - Jan 2021

1 Introduction to the Volatility Anomaly

When people think of option prices, it is almost impossible to disentangle them from the famous model developed in 1973 by Fischer Black, Robert Merton and Myron Scholes (BMS from now onwards) [1].
The model derives the price of European options written on an underlying asset whose price follows a lognormal distribution. It only requires a few inputs: the current asset price (S), the strike price (K), a suitable continuously compounded risk-free rate (r), the option maturity (T), the diffusion parameter or volatility (σ) of the asset return process and, if the underlying pays dividends (or any type of intermediate flows), a continuous dividend yield (q).
Some parameters are straightforward (S can be easily found in the market for quoted companies, K and T are agreed upon by the two counterparties taking part in the trade or set by market conventions for standardized derivatives), other might require a little bit of engineering (r and q are estimated through their discrete-time form and then converted in continuous-time quantities), but the process is quite simple in principle. As to σ, since the BMS formula cannot be inverted analytically to get its explicit expression, it is reverse-engineered from market quotes of option prices through an iterative algorithm.

Two considerations are necessary here. First, the option price is an increasing function of the volatility (and vice-versa), in fact a higher variance of the underlying asset price increases the time value of the derivative without creating additional risk because of the exercise optionality (the maximum loss in a long position is the premium). Second, for each maturity a unique value of the volatility should be found since the parameter is constant and independent of the other parameters according to the BMS framework. However, in market quotes, something different happens: σ is dependent on the strike price and it presents a quite recurrent pattern, the so-called *smiles* or *skews* of the volatility surface. [2] [3] [4]

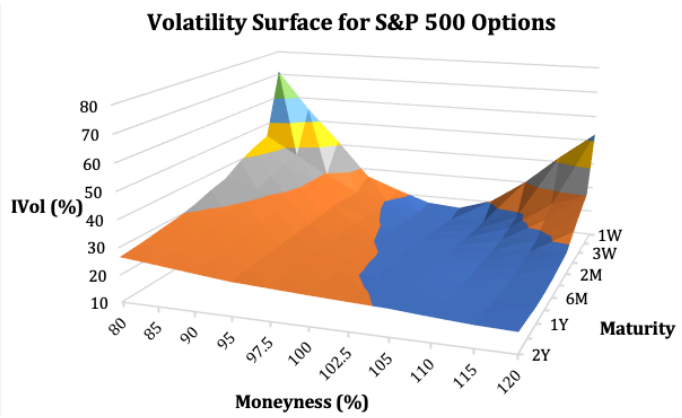

Figure 1: Implied volatility surface for S&P500 options on November 18

This phenomenon is verified in all asset classes options (smiles are frequent for currency and interest rate options, skews for equity index and bond options) and several explanations have been pointed out, without any agreed-upon conclusion reached. Some cite the concern that *lognormal assets* is a too gentle description for some instruments, for example historical distributions of stock (log-)returns usually show negative skewness and positive excess kurtosis. Others mention a sort of fear lingering after the 1987 crash (when this phenomenon started to be verified), for which market participants are willing to inflate the prices of OTM options to avoid bearing large losses. Finally, a few quote the *leverage effect*, which justifies an increase in equity volatility when stocks experience large drops and company leverage increases.

2 Jump Diffusion Model Framework

A different model to price derivatives, introduced by Merton in 1976 [5], can be used to account for discontinuities in the trajectory of the underlying asset. The jump diffusion model (JD from now onwards) is flexible enough to introduce many of the recurrent *stylized facts* for the log-returns distribution, such as non-zero skewness and positive excess kurtosis. The stochastic differential of the underlying asset price can be written in the extended form [6]:

$$dS(t) = S\left(t^{-}\right)\left(\mu dt + \sigma dW(t) + dJ(t)\right) \tag{1}$$

where $J(t) = \sum_{j=1}^{N(t)} (Y_j - 1)$ is a Compound Poisson process. We can therefore find some new parameters with respect to the BMS framework. The first is the intensity λ of the Poisson random variable $N(t)$ which counts the number of jumps occurred until time t (included). Secondly, by assigning lognormal dynamics to Y_j, we define with α and β, respectively, the mean and standard deviation of the log-jump distribution. Finally, by noticing that the jumps unpredictability makes the JD market incomplete, an additional parameter allows to move among the infinite EMMs. Indeed, the price of risk:

$$\theta = \frac{\mu + (E[Y] - 1)\,\phi\lambda^P - r}{\sigma} \tag{2}$$

is an increasing function of ϕ, that we will call risk aversion in the following. By setting up the framework we immediately notice an increase in variability for the underlying stock price:

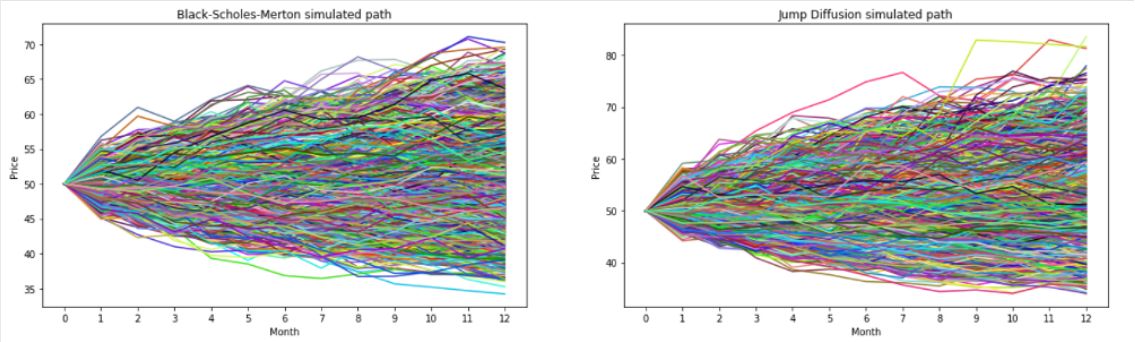

Figure 2: 10,000 simulations for the stock price. $S(0) = 50$, $r = 0.5\%$, $q = 0$, $\sigma = 10\%$, $T = 1$y, $dt = 1$m, $\lambda^Q = 2$, $\alpha = 0.05$, $\beta = 0.001$. Note: The scale of the vertical axes is different

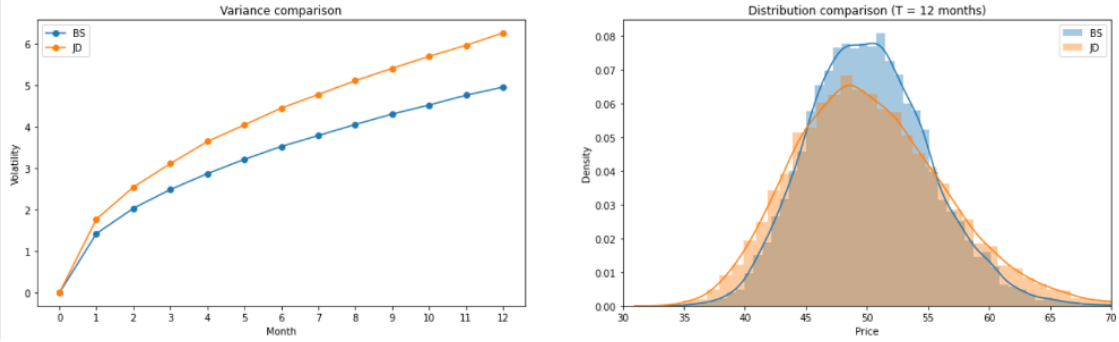

Figure 3: Variance of the stock price at each month and comparison of the terminal price distribution with a standard Normal. Note: The distribution shows fatter tails (and positive skewness since $E[Y] > 1$)

With this backdrop, our aim is to analyze how the volatility smile is affected using a model allowing for a higher variance of the underlying asset price. We think that the opportunity to model risk in addition to the Brownian motion alone can deeply influence the shape of the implied volatility and we intend to perform a sensitivity analysis on each parameter of the JD framework to verify its influence on the resulting pattern.

3 Considerations on the Data

One delicate point is that the pattern of volatility is highly dependent on the liquidity of the underlying market (since it comes directly from market prices). To avoid relying on irregular prices that might have distorted the analysis, we decided to work on plain-vanilla 30-days options written on two famous and widely-available indexes: the S&P 500 and the STOXX Europe 600. In particular, having retrieved the data from Bloomberg on November 18, the options expire on December 18.

It is important to mention that we made sure that the options considered were European (so that our frameworks for both BMS and JD are valid). With regard to the S&P 500, two similar options were found, SPX and SPXW, which differ in terms of reference terminal price (AM vs PM settled) and time of expiry (monthly vs weekly). After researching, we decided to use SPX because of the higher liquidity profile.

Moreover, for each option, we considered the mid price and we double-checked whether the put-call parity held. The answer is, in general, yes, with marginal differences as the skews in the Excel file show. Anyway, our analysis runs the code on both calls and puts to account for non-perfect parity. Given the irrelevant differences in the results, we decided only to report the call options findings.

4 Finding the Implied Volatility

Our analysis relies on the Python language. After having stored the prices in lists, in order to infer the behavior of the IV, we implemented a custom function that, given the option price observed in the market and all the JD formula variables (except for the volatility), returns σ by minimizing the difference between the market and theoretical price.

Our goal is not to merely observe singular IV values for a certain strike, but to understand how it varies for different strikes and parameters. To render the results comparable, we decided to: first fix the JD parameters at some discretionary values ($\alpha = 5\%$, $\beta = 15\%$, $\lambda^P = 2$ and $\phi = 1$), then tweak each one of these, ceteris paribus, for different strikes, plotting the generated implied volatility surface. For instance, in the case of α, the implied volatility was computed for 25 different strikes and 10 different α (equally spacing between -0.2 and 0.2) for a total of 250 IV values.

One criticism for our work is that the parameters are far from what one would get by calibrating the model. Indeed, Andersen and Andreasen [7] performed the calibration on the S&P 500 options using our framework and obtaining α = -88.98%, $\beta = 45.05\%$, $\lambda^P = 8.90\%$ (although imposing $\phi = 1$ and one average volatility instead of the skew), which make intuitive sense since the relevant jumps are more likely to be negative. Moreover, we use the same parameter values for the two sets of options, despite the two markets having quite different historical behavior. However, we only aim at studying the effects of the various parameters on the IV surface, without any calibration to match market values, and proposing different values for the two markets would be totally arbitrary. One consequence is that, in the optimization exercise, the volatility often reaches unreasonable values given our theoretical framework (it is hard to justify $\sigma = 0$ or $\sigma > 100\%$ for a value-weighted market index like the S&P 500).

Regarding the code, we used several well-known Python libraries such as `numpy`, `pandas`, `scipy` and `matplotlib`. We decided to use Python because it allowed us a level of flexibility that we could not find with another language. The final code is quite readable in our opinion. There are several computations involved, since for each graph, as explained above, hundreds of optimizations are involved and each one iterates over the difference between theoretical and market price. This makes the running a bit heavy but, in general, not excessive. In the sections where the optimizer operates, both calls and puts results are computed within the same block for each parameter and the iterations usually take around 90 seconds.

5 Sensitivity Analysis - α

The first parameter that we consider is the expected value of the logarithm of the jump diffusion, α. We let it vary between -20% and 20% and we consider 10 equally spaced values. With $\beta = 15\%$, this means that the expected value of the jump goes from -17.2% to +23.5% of the stock price. As it can be noticed from the following graphs, the implied volatility decays as the strike increases (delineating the classical skew) and as alpha increases.

It seems that negative α in this range do not manage to increase the variance of the underlying enough as to offset the Brownian motion volatility as much as positive values of alpha. Indeed, if α is negative, investors demand high values of volatility in the following days to pay the options at the market prices, while high positive alphas are associated with very low IV levels that in some cases go down to zero. In the latter case α increases the expected value of the jump so much that the jump risk completely dominates the Brownian motion risk. In addition, the effect is much more pronounced in the case of the STOXX Europe, which has a greater variability in the range considered.

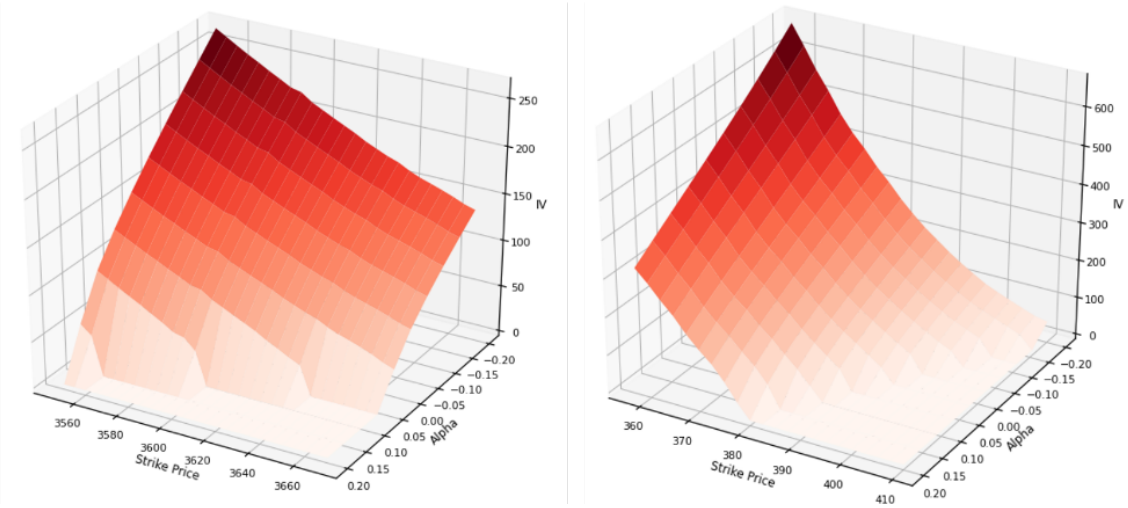

Figure 4: Volatility Surface by changing α for S&P 500 (left) and STOXX Europe 600 (right) options

6 Sensitivity Analysis - β

To complement the analysis on the size of the jump, we performed also an analysis on β. For this parameter, we used different ranges for the two indices. In particular, for the S&P options it is restricted between 5% and 25%, while the upper bound increases until 50% for the STOXX Europe options. This increase is due to the fact that, within the first range, the skew is quite stationary and we are not able to appreciate how much the jump risk affects the volatility estimate.

The graphs in Figure 5 show a clear relationship between beta and the implied volatility: the higher the beta, the lower the IV and vice-versa. This is intuitive: given a strike price, each option price has an associated level of total volatility. While in BMS it is completely attributed to the diffusion risk, in the JD model the volatility is allotted to two sources of risk, one coming from the Brownian motion (represented by the implied volatility) and one coming from the jumps. Thus, once a fixed level of uncertainty is "priced" in the market quotes, when the volatility of the jump model is low, the implied volatility of the Brownian should be high enough to reach this quantity; in the opposite case, when the volatility of the Jump Model is high, the respective implied volatility drops down and, in some cases, it even vanishes (meaning that the jump risk is so high that it fully absorbs the market risk).

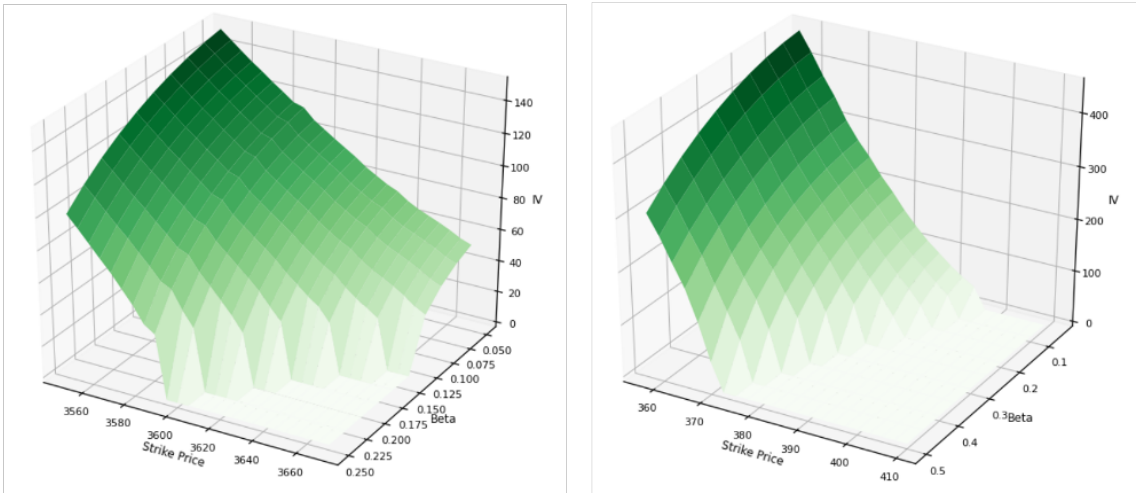

Figure 5: Volatility Surface by changing β for S&P 500 (left) and STOXX Europe 600 (right) options

7 Sensitivity Analysis - The Jump Frequency

We decided to present the sensitivity analysis on λ^P and ϕ jointly since these quantities, respectively the expected number of jumps under the historical measure and the risk aversion, actually combine in one unique relevant parameter, λ^Q. Given λ^P, a higher risk aversion increases the price of risk and translates in a higher number of jumps to be expected in a risk-adjusted world.

Also in this case, the graphs show an inversely proportional relationship between the respective parameter and the implied volatility: the higher λ or ϕ, the lower the IV and vice-versa. These results are not surprising given what has been discussed above. λ represents both the expected value and the variance of the Poisson distribution, namely the random variable that describes the number of jumps, hence a higher value means a higher jump risk which ends up offsetting the risk created by the Brownian motion.

It should be noted that, as with β, we had to modify the ranges for the STOXX Europe options. The surface of the latter, in fact, jumps up to unacceptable values when λ^P is lower than 2 and shows a very relevant sensitivity to the risk aversion.

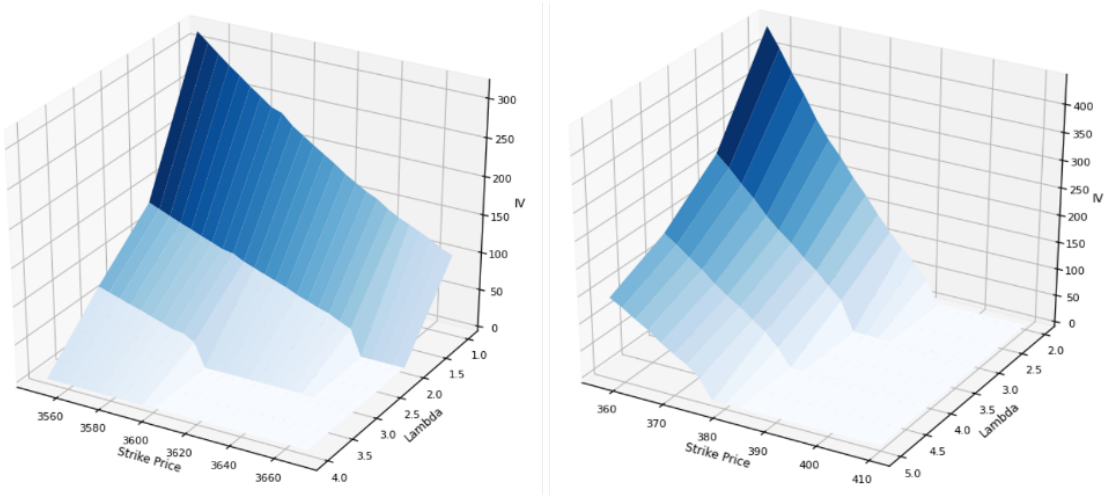

Figure 6: Volatility Surface by changing λ^P for S&P 500 (left) and STOXX Europe 600 (right) options

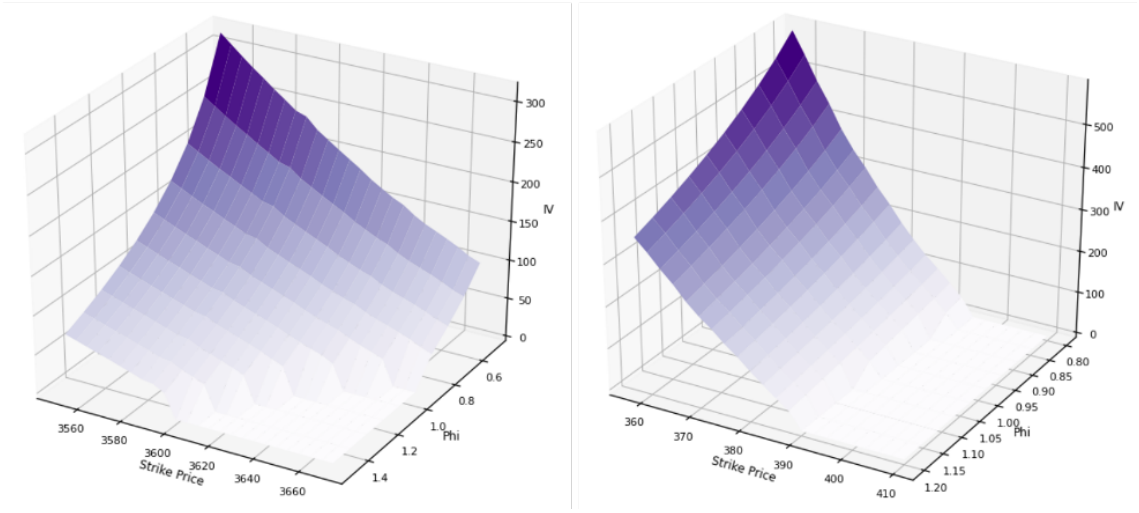

Figure 7: Volatility Surface by changing ϕ for S&P 500 (left) and STOXX Europe 600 (right) options

8 Conclusions and possible extensions

As said in the first sections, our intentions were to study how much each parameter affected the implied volatility surface in the JD model.

We found that, as predicted, parameters that increase the jump risk tend to offset the market risk priced in market quotes. Moreover, all parameters are interconnected and the surface abruptly changes as soon as the ranges are extended and/or the other parameters are modified.

We also saw how the surface changes moving from one market to another. Different index options seem to be sensitive to different parameters. The STOXX Europe 600 options, which have a lower IV than corresponding S&P 500 options when priced through BMS, are much more sensitive to β, λ^P and ϕ. This might come either from the lower volatility to be allotted between the jump and the market risk or from the lower liquidity that these options show when compared to the S&P ones.

An interesting extension to our work would be to actually calibrate the models and to check how the implied volatility changes when the ranges include the actual estimate of the parameters.

References

[1] F. Black and M. Scholes. The Pricing of Options and Corporate Liabilities. *The Journal of Political Economy*, 81(3):637–654, 1973.

[2] S. Ross. What is a volatility smile? `https://www.investopedia.com/ask/answers/012015/what-volatility-smile.asp`, 2019.

[3] W. Kenton. Volatility Skew Definition. `https://www.investopedia.com/terms/v/volatility-skew.asp`, 2020.

[4] M. Cory. Volatility Smile Definition and Uses. . `https://www.investopedia.com/terms/v/volatilitysmile.asp`, 2019.

[5] R. Merton. Option pricing when underlying stock returns are discontinuous. *Journal of Financial Economics*, 3(1-2):125–144, 1976.

[6] A. Battauz. Topics in Quantitative Finance, Lecture Notes. *Bocconi University*, 2020.

[7] L. Andersen and J. Andreasen. Jump-Diffusion Processes: Volatility Smile Fitting and Numerical Methods for Option Pricing. *Review of Derivatives Research*, 4:231–262, 2000.

Appendix

```
def BS_simulate(S, r, q, sigma, dt):
  '''
  Simulate the stock price according to Black-Scholes model
  in the time interval dt

  S: starting stock price
  r: annualized continuous risk-free interest rate
  q: continuos dividend yield of the underlying asset
  sigma: annualized volatility of the underlying asset
  dt: time interval
  '''
  return S * np.exp((r - q - (sigma**2) / 2) * dt + sigma * np.sqrt(dt) * np.
    random.randn())

def JDM_simulate(S, r, q, sigma, dt, l, a, b):
  '''
  Simulate the stock price according to Jump diffusion model
  in the time interval dt having as log-jumps' distribution
  a Normal with mean a and volatility b

  S: starting stock price
  r: annualized continuous risk-free interest rate
  q: continuos dividend yield of the underlying asset
  sigma: annualized volatility of the underlying asset
  dt: time interval
  l: lambda parameter for the Poisson distribution (Merton assumption of unit phi
    is assumed)
  a: mean of the logarithm of the jump distribution
  b: standard deviation of the logarithm of the jump distribution
  '''
  N = np.random.poisson(l * dt)
  if N > 0:
    Y = np.exp(np.random.randn() * b * np.sqrt(N) + a * N)
  else:
    Y = 1
  m = np.exp(a + 0.5 * b**2) - 1
  return BS_simulate(S, r - m * l, q, sigma, dt) * Y

def path_simulations(simulations, T, intervals, model, S, **kwargs):
  '''
  Simulate the paths of a stock price according to different models
  and save them in a dataframe

  simulations: number of simulations you want to perform
  T: time horizon
  intervals: number of intermediate steps at which you want to simulate your stock
  model: BS_simulate or JDM_simulate
  S: initial stock price

  **kwargs: all other parameters that you need to furnish to your model;
  Allowed parameters:
```

```
    r: annualized continuous risk-free interest rate
    q: continuos dividend yield of the underlying asset
    sigma: annualized volatility of the underlying asset
    dt: time interval
    l: lambda parameter for the Poisson distribution
    a: mean of the logarithm of the jump distribution
    b: standard deviation of the logarithm of the jump distribution
  '''

  df = pd.DataFrame(columns = range(0, intervals + 1))
  df.loc[:, 0] = [S] * simulations
  for j in range(simulations):
    for i in range(1, intervals + 1):
      df.loc[j, i] = model(S = df.loc[j, i - 1], dt = T / intervals, **kwargs)
  return df
```

Listing 1: Code used for the simulation images

```

def Price_Option_BMS (option_type, S, K, r, q, sigma, T):
    '''
    returns the price of a plain vanilla European option on assets
    paying a continuos dividend yield using the Black-Merton-Scholes model

    option_type: call/put option
    S: spot price of the underlying asset
    K: strike price
    r: annualized continuous risk-free interest rate
    q: continuos dividend yield of the underlying asset
    sigma: annualized volatility of the underlying asset
    T: annualized time to expiration
    '''
    d1 = (np.log(S / K) + (r - q + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    call_price = S * np.exp(-q * T) * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf
    (d2)

    if option_type.upper().startswith('C'): # call option
      return call_price
    elif option_type.upper().startswith('P'): # put option
      put_price = K * np.exp(-r * T) - S * np.exp(-q * T) + call_price # put-call
    parity
      return put_price

def Price_Option_JD (option_type, S, K, r, q, sigma, T, lambdaP, a, b, phi = 1, N
    = 50):
  '''
    returns the price of a plain vanilla European option on assets
    paying a continuos dividend yield using the Jump Diffusion model;
    a plurality of risk-neutral are accepted by setting the risk aversion
    parameter;
    Jumps are modelled as lognormal random variables

    option_type: call/put option
```

```
    S: spot price of the underlying asset
    K: strike price
    r: annualized continuous risk-free interest rate
    q: continuos dividend yield of the underlying asset
    sigma: annualized volatility of the underlying asset
    T: annualized time to expiration
    lambdaP: lambda parameter for the Poisson distribution under the historical probability measure
    a: mean of the logarithm of the jump distribution
    b: standard deviation of the logarithm of the jump distribution
    phi: (optional) risk aversion parameter, default value is 1
    N: (optional) number of terms in the summation of BMS options (i.e. number of jumps to be considered), default value is 50
    '''
  # expected value of one jump under the risk-neutral probability measure:
  miY = np.exp(a + 0.5 * b**2)
  m = miY - 1
  # lambda parameter for the Poisson distribution under the risk-neutral probability measure:
  lambdaQ = phi * lambdaP
  lambda_prime = lambdaQ * miY

  # computation of the probability-weighted summation of BMS options
  call_price = 0
  for n in range(1, N + 1):
    r_n = r - m * lambdaQ + n * np.log(miY) / T
    sigma_n =np.sqrt((sigma**2 * T + n * b**2) / T)
    call_price += np.exp(-lambda_prime * T) * (lambda_prime * T)**n / math.factorial(n) * \
    Price_Option_BMS('Call', S, K, r_n, q, sigma_n, T)

  if option_type.upper().startswith('C'): # call option
    return call_price
  elif option_type.upper().startswith('P'): # put option
    put_price = K * np.exp(-r * T) - S * np.exp(-q * T) + call_price # put-call parity
    return put_price
```

Listing 2: Prices of call and put option under original Black and Scholes or Jump Diffusion frameworks

```
def find_IV_JD (option_type, S, K, r, q, T, lambdaP, a, b, actual_price, phi = 1, N = 50):  '''
  Find the implied volatility with the JD framework

  option_type: (str) 'C' for call option and 'P' for put
  S: spot price of the underlying asset
  K: strike price
  r: annualized risk-free interest rate
  q: continuos dividend yield of the underlying asset
  T: annualized time to expiration
  lambdaP: lambda parameter for the Poisson distribution under the historical probability measure
  a: mean of the logarithm of the jump distribution
  b: standard deviation of the logarithm of the jump distribution
```

```
  phi: (optional) risk aversion parameter, default value is 1
  N: (optional) number of terms in the summation of BMS options (i.e. number of
    jumps to be considered), default value is 50
  '''
  func = lambda sigma: Price_Option_JD (option_type, S, K, r, q, sigma, T, lambdaP
    , a, b, phi, N)
  f = lambda sigma:(func(sigma) - actual_price)**2
  bnds = [(0, None)]
  cons = {'type': 'ineq', 'fun': lambda x:  x}
  result = opt.minimize(f, 0.5, method = 'SLSQP', bounds = bnds, constraints =
    cons)
  return result
```

Listing 3: Optimization algorithm

```
# Vector of implied volatilities for the call options:
jiv_call = []
# Vector of implied volatilities for the put options:
jiv_put = []
# Vector of strike prices:
strikes = []
# Vector of means for the log jump i.e. a:
alphas = []

for K, actual_call_price, actual_put_price in zip(range(3550, 3675, 5),
    call_prices, put_prices):
  for a in np.linspace(-0.2, 0.2, 10):
    update_call = find_IV_JD (option_type = 'C', S = 3609.53, K = K, r = 0.0023, q
     = 0.0144, T = 30/365, lambdaP = 2, a = a, b = 0.15, actual_price =
    actual_call_price).x
    update_put = find_IV_JD (option_type = 'P', S = 3609.53, K = K, r = 0.0023, q
    = 0.0144, T = 30/365, lambdaP = 2, a = a, b = 0.15, actual_price =
    actual_put_price).x
    jiv_call.append(update_call * 100)
    jiv_put.append(update_put * 100)
    strikes.append(K)
    alphas.append(a)
```

Listing 4: Sample code used to do sensitivity analysis (alpha)