Option Pricing: The Binomial Model & The Black-Scholes Model With Monte Carlo Simulation & Variance Reduction Techniques

The Notebook comprises of the following:

- The Binomial Options Pricing Model Converges to Black-Scholes as T Tends to Infinite.
- The Black-Scholes Options Pricing Model With Monte Carlo Simulation.
- Options Pricing With Variance Reduction Techniques :
    - Control Variates.
    - Antithetic Variates.

We assume the underlying asset of the option is a stock.

First, lets import all the relevant Python packages required.

In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import math
from scipy.stats import norm
```

Now we create a function for the Binomial model formula for a European call option which is given by:

$$C_0 = \frac{1}{B_N}\sum_{k=0}^{N}\binom{N}{k}\rho_N^k(1-\rho_N)^{N-k}(S_0u_N^kd_N^N-K)^+$$

Where:

- B_N is the
- ρ is the risk-neutral probability: $\rho = \frac{1+r-d}{u-d}$
- N is the number of periods.
- k is the particular observation.
- S_0 is the current stock price.
- K is the exercise price of the call option.
- u is the level of upstate.
- d is the level of downstate.

In [2]:
```python
def bin_call(N, S0, K, u, d, r):
    rho = (1 + r - d) / (u - d)
    price = 0
    for k in range(0, N+1):
        stock_price = S0 * (u ** k) * (d ** (N - k))
        price = price + math.comb(N, k) * (rho ** k) * ((1 - rho) ** (N - k)) * np.maximum(stock_price - K, 0)
        final_result = price / ((1 + r) ** N)
    return final_result

print(f'Binomial Option Price: ${bin_call(N=10, S0=100, K=100, u=2, d=0.5, r=0.04):.3f}')
```

```
Binomial Option Price: $76.378
```

Now before we compare the two models we will need to approximate the Black-Scholes option price with specific risk-neutral probabilities using the above function in order to a draw a conclusion.

In [3]:
```python
def bl_sc_call_approx(N, S0, K, T, sigma, r):
    u_bl_sc = np.exp(sigma * np.sqrt(T / N))
    d_bl_sc = 1 / u_bl_sc
    r_bl_sc = np.exp(r * T / N) - 1
    price = bin_call(N, S0, K, u_bl_sc, d_bl_sc, r_bl_sc)
    return price
```

Now lets create a function that enables us to compute the Black-Scholes European Call Option Prices using the following formula:

$$C_0 = S_0\Phi(d1) - Ke^{-rT}\Phi(d2)$$

Where

- C_0 is the European call option price at time-0.
- S_0 is the current price of the underlying, in our case lets take a stock.
- K is the exercise price of the option.
- r is the risk-free interest rate.
- T is the time to maturity
- Φ is the cumulative distribution function of the Normal distribution.
- $d1 = \frac{\frac{logS_0}{K}+(r+0.5\sigma^2)T}{\sigma\sqrt{T}}$
- $d2 = d1 - \sigma\sqrt{T}$

In [4]:
```python
def bl_sc_call(S0, K, T, sigma, r):
    d1 = (np.log(S0/K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    term_1 = S0 * norm.cdf(d1, loc=0, scale=1)
    term_2 = K * np.exp(-r * T) * norm.cdf(d2, loc=0, scale=1)
    price = term_1 - term_2
    return price
```

In [5]:
```python
approx_price = bl_sc_call_approx(N=1000, S0=100, K=100, T=1, sigma=0.25, r=0.05)
theoretical_price = bl_sc_call(S0=100, K=100, T=1, sigma=0.25, r=0.05)
print(f'Approximate Price: ${approx_price:.3f}')
print(f'Theoretical Price: ${theoretical_price:.3f}')
```

```
Approximate Price: $12.334
Theoretical Price: $12.336
```

As we can note above that the approximation through the N-period binomial model yields results very close to the Black-Scholes model.

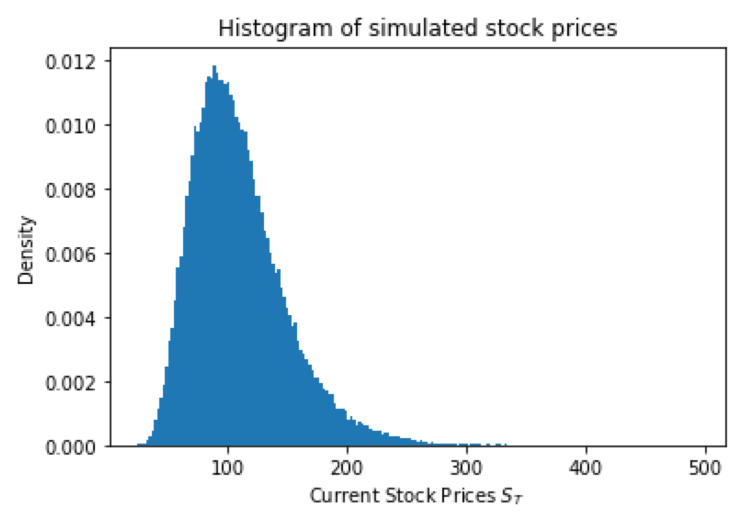
Now lets create a function that generates samples of stock prices in the Black-Scholes model (i.e. simulate) and visualise it to see its distribution which should result a standard normal distribution.

In [6]:
```python
rng = np.random.default_rng(1)

def bl_sc_samples_generator(rng, sample_size, S0, T, sigma, r):
    #Generator of random numbers with standard normal distribution.
    samples = rng.standard_normal(size=sample_size)
    #Black-Scholes formula application
    term_1 = (r - 0.5 * sigma ** 2) * T
    term_2 = sigma * np.sqrt(T) * samples
    stock_prices = S0 * np.exp(term_1 + term_2)
    return stock_prices
```

In [7]:
```python
example_2 = bl_sc_samples_generator(rng, sample_size=100000, S0=100, T=2, sigma=0.25, r=0.05)

fig, ax = plt.subplots()
ax.hist(example_2, bins=200, density=True)
ax.set_title("Histogram of simulated stock prices")
ax.set_xlabel("Current Stock Prices $S_T$")
ax.set_ylabel("Density");
```



Now lets use the sample generator fucntion that we created and implement it in a new function which computes the Monte Carlo estimator of the stock prices and depict its 95% confidence interval, so that we can make statistical inferences.

In [8]:
```python
#epsilon is the confidence interval %

def bl_sc_mc_estimator(rng, sample_size, epsilon, S0, K, T, sigma, r):
    samples = bl_sc_samples_generator(rng, sample_size, S0, T, sigma, r)
    payoffs = np.maximum(samples - K, 0)
    discounted_payoffs = np.exp(- r * T) * payoffs
    price = np.mean(discounted_payoffs)
    sd_rv = np.std(discounted_payoffs, ddof=1)
    sd_mc_estimator = sd_rv / np.sqrt(sample_size)
    normalised_epsilon = norm.ppf(1 - epsilon * 0.5)
    left_ci = price - normalised_epsilon * sd_mc_estimator
    right_ci = price + normalised_epsilon * sd_mc_estimator
    return price, sd_mc_estimator, left_ci, right_ci
```

Now lets define the model parameters and compare the theoretical price from the Black-Scholes model and the Monte Carlo estimator.

In [9]:
```python
sample_size = 1000000
epsilon = 0.05
S0 = 100
K = 100
T = 2
sigma = 0.25
r = 0.05
```

In [24]:
```python
theoretical_price = bl_sc_call(S0, K, T, sigma, r)
print(f'Theoretical Price: ${theoretical_price:.3f}')
mc_results = bl_sc_mc_estimator(rng, sample_size, epsilon, S0, K, T, sigma, r)
print(f'Monte Carlo Price: ${mc_results[0]:.3f} & Standard Deviation: {mc_results[1]:.3f}, \
Confidence Interval: ({mc_results[2]:.3f},{mc_results[3]:.3f})')
```

```
Theoretical Price: $18.647
Monte Carlo Price: $18.670 & Standard Deviation: 0.029, Confidence Interval: (18.614,18.726)
```

Above we observe that the true value does indeed lie in the 95% confidence interval of the Monte Carlo estimator thus we can conclude that options pricing through Monte Carlo simulation provides a good estimate of prices computed via the Black-Scholes model.

Now lets look at control variates variance reduction technique within Monte Carlo simulation for the Black-Scholes option pricing model.

Control variates method is used when the goal is to try to simulate the expected value of a random variable, X.Then a second random variable, Y, is introduced for which the expected value is known.Then we try to maximise the correlation between the two random variables such that the variance of the main random variable, X, is reduced thus improving the accuracy of the estimator.In reality, finding such highly correlated variables can be difficult in the context of financial securities.

Lets create a function for the control variate estimator using the functions that we have already created above and depict its standard deviation, 95% confidence intervals and the correlation between the two random variables.Here we take the discounted stock prices as control as they will be highly correlated with the discounted payoffs since there is a linear relationship.

In [11]:
```python
def cv_bl_sc_call(rng, sample_size, epsilon, S0, K, T, sigma, r):
    stock_price = bl_sc_samples_generator(rng, sample_size, S0, T, sigma, r)
    payoffs = np.maximum(stock_price - K, 0)
    discounted_payoffs = np.exp(-r * T) * payoffs
    control = np.exp(-r * T) * stock_price
    minimised_variance_Y = np.cov(control, discounted_payoffs, ddof=1)[0, 1] / np.var(control, ddof=1)
    term = discounted_payoffs - minimised_variance_Y * (control - S0)
    price = np.mean(term)
    sd_rv = np.std(term, ddof=1)
    sd_cv = sd_rv / np.sqrt(sample_size)
    normalised_epsilon = norm.ppf(1 - epsilon * 0.5)
    left_ci = price - normalised_epsilon * sd_cv
    right_ci = price + normalised_epsilon * sd_cv
    cov_XY = np.cov(control, discounted_payoffs, ddof=1)[0, 1]
    var_X = np.var(control, ddof=1)
    var_Y = np.var(discounted_payoffs)
    corr = (cov_XY ** 2) / (var_X * var_Y)
    return price, sd_cv, left_ci, right_ci, corr
```

In [12]:
```python
cv_results = cv_bl_sc_call(rng, sample_size=1000000, epsilon=0.05, S0=100, K=100, T=2, sigma=0.25, r=0.05)
print(f'Control Variate Price: ${cv_results[0]:.3f} & Standard Deviation: {cv_results[1]:.3f}, \
Confidence Interval: ({cv_results[2]:.3f},{cv_results[3]:.3f}), Correlation: {cv_results[4]:.3f}')
```

```
Control Variate Price: $18.637 & Standard Deviation: 0.009, Confidence Interval: (18.619,18.655), Correlation: 0.893
```

As we can see above the standard deviation has reduced significantly compared to the Monte Carlo simulation without variance reduction, 0.09 to 0.029.We also notice that the control random variable that we took is higly correlated with the target random variable.Moreover, the confidence interval has also significantly been narrowed down and does contain the expected value (i.e. the theoretical price).

Antithetic variates is another technique which enables us to reduce variance.

Antithetic variates is similar to control variates where it tries to utilise the negative correlation between random variables in order to reduce variance.It employs inverse transform method on general distribution to generate antithetic pairs which enables us to find the estimator with the reduced variance.

Lets consider the following antithetic pair: $(X, -X)$ where $X \sim \mathcal{N}(0,1)$ Therefore our stock price pair would be: $(S^{(1)}, S^{(2)})$ and equations would be:

$$S^{(1)} = S_0exp((r-0.5\sigma^2)T + \sigma\sqrt{T}X$$

$$S^{(2)} = S_0exp((r-0.5\sigma^2)T + \sigma\sqrt{T}(-X)$$

We will need to create a new stock price sample generator which satisfies the above conditions.

In [16]:
```python
def av_bl_sc_samples_generator(rng, half_sample_size, S0, T, sigma, r):
    normal_1 = rng.standard_normal(half_sample_size)
    normal_2 = -normal_1
    term_1 = (r - 0.5 * sigma ** 2) * T
    term_2 = sigma * np.sqrt(T) * normal_1
    term_3 = sigma * np.sqrt(T) * normal_2
    stock_price_1 = S0 * np.exp(term_1 + term_2)
    stock_price_2 = S0 * np.exp(term_1 + term_3)
    combined_stock_prices = np.concatenate((stock_price_1, stock_price_2))
    return stock_price_1, stock_price_2, combined_stock_prices
```

In [19]:
```python
def av_bl_sc_call(rng, half_sample_size, epsilon, S0, K, T, sigma, r):
    stock_prices = av_bl_sc_samples_generator(rng, half_sample_size, S0, T, sigma, r)
    payoffs_1 = np.maximum(stock_prices[0] - K, 0)
    payoffs_2 = np.maximum(stock_prices[1] - K, 0)
    discounted_payoffs_1 = np.exp(-r * T) * payoffs_1
    discounted_payoffs_2 = np.exp(-r * T) * payoffs_2
    cov_12 = np.cov(discounted_payoffs_1, discounted_payoffs_2, ddof=1)[0, 1]
    reduction = cov_12 / (2 * half_sample_size)
    combined_discounted_payoffs = np.concatenate((discounted_payoffs_1, discounted_payoffs_2))
    price = np.mean(combined_discounted_payoffs)
    sd_rv = np.std(combined_discounted_payoffs, ddof=1)
    sd_av = sd_rv / np.sqrt(2 * half_sample_size)
    normalised_epsilon = norm.ppf(1 - epsilon * 0.5)
    left_ci = price - normalised_epsilon * sd_av
    right_ci = price + normalised_epsilon * sd_av
    return price, sd_av, left_ci, right_ci, reduction
```

In [23]:
```python
av_results = av_bl_sc_call(rng, half_sample_size=500000, epsilon=0.05, S0=100, K=100, T=2, sigma=0.25, r=0.05)
print(f'Antithetic Variate Price: ${av_results[0]:.3f} & Standard Deviation: {av_results[1]:.3f}, \
Confidence Interval: ({av_results[2]:.3f},{av_results[3]:.3f}), Reduction: {av_results[4]:.3f}')
```

```
Antithetic Variate Price: $18.702 & Standard Deviation: 0.029, Confidence Interval: (18.646,18.758), Reduction: -0.000
```

Above we observe that the standard deviation has remained the same at 0.029 which is not a reduction compared to the significant reduction in the control variate technique.However, the antithetic variate estimator is very close to the theoretical price as compared to the the control variate estimator.

This concludes the notebook.