CQF Exam Two Report - Harrison King

June 2023 Cohort

1. Financial and Numerical Outline

The finance problem is to price Asian and lookback options following the risk-neutral Black-Scholes-Merton framework, modelled using the Euler-Maruyama scheme to simulate the underlying asset prices. We will use a Monte Carlo numercial procedure to price the options due to the strong path-dependency of both types of options. As the underlying has the martingale property we will then use the expected value of the discounted payoff under the risk-neutral density $\mathbb{Q}$ to price the options.

$$V(S,t) = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}\left[\text{ Payoff }\left(S_T\right)\right]$$

1.1 The Euler-Maruyama Scheme

In this report the Euler-Maruyama scheme is used to discretise the underlying asset price. This scheme is a way of approximating numerical solutions to stochastic differential equations (SDEs).

Assuming that the underlying asset price follows a Geometric Brownian Motion (GBM) process, then the log of the returns are normally distributed and the underlying follows the Markov property. Therefore the SDE for the asset price is given by the risk-neutral random walk of the asset price, S_t:

$$dS_t = rS_t dt + \sigma S_t dW_t \tag{1}$$

where r is the risk-free interest rate, σ is the volatility of the asset price and W_t is a Wiener process and is a Normally distributed random variable such that $dW(t) \sim N(0, dt)$.

Dividing through by S_t we get:

$$\frac{dS_t}{S_t} = rdt + \sigma dW_t \tag{2}$$

The solution S_t can be found by applying Itô's lemma to the SDE. From Itô where $F(S) = log(S)$ we have:

$$dF = \frac{dF}{dS_t}dS + \frac{1}{2}\sigma^2 S_t^2 \frac{d^2F}{dS_t^2}dt = \frac{1}{S_t}(rS_t dt + \sigma S dW_t) \tag{3}$$

$$dF = (r - \frac{1}{2}\sigma^2)dt + \sigma dW_t \tag{4}$$

Therefore, by substituting $F(S) = log(S)$ into $\text{Equation } (4)$ for the lognormal random walk we can write the risk-neutral SDE for the asset price S_t as:

$$d(logS_t) = (r - \frac{1}{2}\sigma^2)dt + \sigma dW_t \tag{5}$$

Integrating $\text{Equation } (5)$ and taking the exponential gives:

$$S_t = S_0 \exp\left(\left(r - \frac{1}{2}\sigma^2\right)t + \sigma \int_0^t dW_t\right) \tag{6}$$

With ϕ_i as a standard Normally distriubted variable, we can then can express the risk-neutral asset price using the Euler-Maruyama method over a time step δt as:

$$S_{t+\delta t} = S_t \exp\left(\left(r - \frac{1}{2}\sigma^2\right)\delta t + \sigma\sqrt{\delta}t\phi_i\right) \tag{7}$$

We will use the Euler-Maruyama scheme derived in $\text{Equation } (7)$ to simulate the underlying asset price paths for Asian and lookback options. We will use the Monte Carlo method to price the options as they are path-dependent options with three dimensions, this is discussed in more detail in the following sections.

1.2 Asian Options

1.2.1 Financial Outline of Asian Options

Asian options are a type of exotic option that have a payoff which depends on the average price of the underlying asset over some period before expiry. They are commonly traded for currencies, commodities, interest rates and energy markets.

They are the strongly path dependent options because their value prior to expiry depends on the average to date of the asset rather than just the final price. This means that unlike vanilla options with a dimensionality of two (two independent variables, S and t) the Asian option is a three-dimensional problem and the differential equation has three independent variables, the underlying asset price, S, time, t, and the average price of the underlying asset over the life of the option, $A(0,T)$.

Payoff Types of an Asian Option

There are two classifications of the payoff type of Asian options:

- **Fixed strike or average rate**, where the average price is used in place of the underlying asset price in the payoff function.
- **Floating strike or average strike**, where the average price is used in place of the strike price in the payoff function.

The call, C_T, and put, P_T, payoffs for a **fixed strike** Asian option are given by:

$$C_T = \max\left(A(0,T) - E, 0\right) \tag{8}$$

$$P_T = \max\left(E - A(0,T), 0\right) \tag{9}$$

And the call, C_T, and put, P_T, payoffs for a **floating strike** Asian option are given by:

$$C_T = \max\left(S(T) - A(0,T), 0\right) \tag{10}$$

$$P_T = \max\left(A(0,T) - S(T), 0\right) \tag{11}$$

where E is the strike price and T is the time to maturity.

1.2.2 Numerical Approach of Pricing Asian Options

Arithmetic and Geometric Asian Options

The average price, $A(0,T)$, can be computed in several ways, the most common being the arithmetic and geometric average. The arithmetic average is the sum of the asset prices, equally weighted, divided by the total number of prices used. The geometric average on the other hand is the exponential of the sum of all the logarithms of the constituent prices, equally weighted, divided by the total number of price used. Continuously sampled geometric Asian options are easy to price as there is a closed form analytical solution using the Black-Scholes formula. This is because the geometric average of a lognormally distributed underlying asset has a lognormal distribution.

Conversely, arithmetic Asian options are difficult to price as the arithmetic average of a lognormally distributed underlying asset does not have a lognormal distribution. This means that we cannot use the Black-Scholes formula to price the option and instead we must use a Monte Carlo approach for pricing. As the Monte Carlo scheme of pricing exotic options is the focus of this report, we will focus on arithmetic Asian options.

Discrete and Continuous Sampled Averages

The data could be continously sampled or sampled at discrete times to calculate averages. With the former, using every realised price over a finite time period, the sums calculated in the average become integrals of the asset price over the averaging period. This gives a continuously sampled average. With the latter and more common sampling approach, the averaging period is divided into N discrete time intervals beginning at time $t = 0$ and ending at maturity, T. The average is calculated using the asset price at the end of each interval (for example, using the closing price of the asset at the end of each day). This gives a discretely sampled average. The discretely sampled average is more common outside of theoretical contexts as it is more practical and more realistic. This will be the main approach taken in this report.

The discretely sampled arithmetic average of the asset price is defined by:

$$A(0,T) = \frac{1}{N}\sum_{i=1}^{N} S(t_i) \tag{12}$$

And the discretely sampled geometric mean is defined by:

$$A(0,T) = \exp\left(\frac{1}{N}\sum_{i=1}^{N}\ln S(t_i)\right) \tag{13}$$

Monte Carlo Pricing of Asian Options

Monte Carlo Simulations are used to price Asian options as they are strongly path-dependent options with three dimensions.

Taking the fixed strike arithemetic average Asian option values from the call and put payoffs in $\text{Equation } (8)$ and $\text{Equation } (9)$, we can price the options using the Monte Carlo method. The Monte Carlo method is then:

1. Simulate the risk-neutral random walk over the time horizon, starting at $S(0)$ and ending at $S(T)$. This is done using the Euler-Maruyama scheme and gives one realisation of the underlying price path.
2. Perform many more realisations to generate a total of 10,000 random realisations over the time horizon.
3. Calculate the payoff for each realisation of the underlying price path using the discretely sampled arithmetic average of the asset price over the life of the option, $A(0,T)$, and the strike price, E given by $\text{Equation } (12)$.
4. Calculate the average or *expected* payoff over all the realisations.
5. Present value the expected payoff using the risk-free interest rate, r, and the time to maturity, T.

The expected present value of a fixed strike arithmetic Asian call, C, and put, P, option with time to expiry, T, is therefore:

$$C = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}\left[\max\left(A(0,T) - E, 0\right)\right] \tag{14}$$

$$P = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}\left[\max\left(E - A(0,T), 0\right)\right] \tag{15}$$

where $\mathbb{E}^{\mathbb{Q}}$ is the expectation under the risk-neutral measure.

1.3 Lookback Options

1.3.1 Financial Outline of Lookback Options

Lookback options are a type of exotic option that have a payoff which depends on the maximum or minimum price of the underlying asset over some period before expiry. Since they have such extreme payoffs, they are often expensive contracts.

Lookback options are strongly path dependent options because their value prior to expiry depends on the maximum or minimum to date of the asset rather than just the final price. This means that unlike vanilla options with a dimensionality of two (two independent variables, S and t), and the same as Asian options, the lookback option is three-dimensional due to having three independent variables, the underlying asset price, S, the time, t, and the maximum or minimum price of the underlying asset over the life of the option, $M(0,T)$.

Payoff Types of a Lookback Option

Similarly to Asian options, there are two classifications of the payoff type of lookback options:

- **Fixed strike or average rate**, where the maximum or minimum price is used in place of the underlying asset price in the payoff function.
- **Floating strike or average strike**, where the maximum or minimum price is used in place of the strike price in the payoff function.

The call, C_T, and put, P_T, payoffs for a **fixed strike** lookback option are given by:

$$C_T = \max\left(M(0,T) - E, 0\right) \tag{16}$$

$$P_T = \max\left(E - M(0,T), 0\right) \tag{17}$$

And the call, C_T, and put, P_T, payoffs for a **floating strike** lookback option are given by:

$$C_T = \max\left(S(T) - M(0,T), 0\right) \tag{18}$$

$$P_T = \max\left(M(0,T) - S(T), 0\right) \tag{19}$$

where E is the strike price and T is the time to maturity.

1.3.2 Numerical Approach of Pricing Lookback Options

Discrete and Continuous Measurement of the Maximum

Much like Asian options, the path dependent variable ($M(0,T)$) can be calculated with continous or discrete sampling. Focussing just on the maximum lookback option with the former, using every realised price over a finite time period, the maximum price becomes the maximum of the asset price over the prescribed period giving a continously sampled maximum. With this sampling methodology the realised asset price is always less than or equal to the maximum. This theoretical approach allows for a closed form analytical solution using the Black-Scholes formula, however as with Asian options continous monitoring is less practical than discrete.

The discretly sampled approach of dividing the prescribed period into N discrete time intervals beginning at time $t = 0$ and ending at maturity, T is commonly used in practise. The discretly sampled maximum is calculated using the asset price at the end of each interval (for example, using the closing price of the asset at the end of each day). With this methodology, the asset price can exceed the discretely sampled maximum, $M(0,T)$. Additionally, the maximum is much less frequently increased across the prescribed period, therefore as well as being more practical than continous sampling, the discretly sampled maximum results in a lower option price and is used to decrease the value of a contract.

The discretely sampled maximum and minimum of the asset price are defined by:

$$M(0,T) = \max\left(S(t_1), S(t_2), \ldots, S(t_N)\right) \tag{20}$$

$$M(0,T) = \min\left(S(t_1), S(t_2), \ldots, S(t_N)\right) \tag{21}$$

Monte Carlo Pricing of Lookback Options

The Monte Carlo Pricing of lookback options is very similar to that of Asian options. The Euler-Maruyama scheme assuming a GBM process is used to simulate the underlying asset price as derived in $\text{Equation } (7)$.

Taking the fixed strike maximum and minimum lookback option values from the call and put payoffs in $\text{Equation } (16)$ and $\text{Equation } (17)$, we can price the options using the Monte Carlo method. The Monte Carlo method is then:

1. Simulate the risk-neutral random walk over the time horizon, starting at $S(0)$ and ending at $S(T)$. This is done using the Euler-Maruyama scheme and gives one realisation of the underlying price path.
2. Perform many more realisations to generate a total of 10,000 random realisations over the time horizon.
3. Calculate the payoff for each realisation of the underlying price path using the maximum or minimum of the asset price over the life of the option, $M(0,T)$, and the strike price, E given by $\text{Equation } (20)$ and $\text{Equation } (21)$.
4. Calculate the average or *expected* payoff over all the realisations.
5. Present value the expected payoff using the risk-free interest rate, r, and the time to maturity, T.

The expected present value of a fixed strike maximum or minimum lookback call, C, and put, P, option with time to expiry, T, is therefore:

$$C = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}\left[\max\left(M(0,T) - E, 0\right)\right] \tag{22}$$

$$P = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}\left[\max\left(E - M(0,T), 0\right)\right] \tag{23}$$

1.4 Numerical Errors

For path dependent options such as Asian and lookback options where the time step is δt, we may introduce errors in the order of magnitude $O(\delta t)$ due to the discrete approximation of the continuous underlying asset price. Additionally, the error of simulating N finite paths of a possible infinite number of paths is $O\left(\frac{1}{\sqrt{N}}\right)$.

We can then say that the total time taken to estimate the price of an option to a given accuracy using the Monte Carlo method is $O\left(\frac{N}{\delta t}\right)$.

The error in the price is then given either the greater of the error due to the discreteness of the time step or the error due to the finite number of paths. This is given by:

$$O\left(max\left(\delta t, \frac{1}{\sqrt{N}}\right)\right) \tag{24}$$

In this report we will use a time step of $\delta t = 1/252$ representing the number of trading days in a year, and $N = 10,000$ paths. This gives an error of:

$$O\left(max\left(1/252, \frac{1}{\sqrt{10,000}}\right)\right) = O\left(max\left(1/252, 0.01\right)\right) = O\left(0.01\right)$$

.

1.4.1 Standard Error

Looking at a call option with payoff C, the standard error of the expected payoff, $SE(\mathbb{E}^{\mathbb{Q}}[C])$, is the standard deviation of the payoffs, $\sigma(C)$, divided by the square root of the number of paths, $\sqrt{N}$, assuming the expected payoff is normally distributed:

$$SE(\mathbb{E}^{\mathbb{Q}}[C]) = \frac{\sigma(C)}{\sqrt{N}} \tag{25}$$

2. Implementation and Results

2.1 Monte Carlo Simulation Class

As we assume that the underlying asset price for both Asian and lookback options both follow a Geoemtric Brownian Motion (GBM) process we will define a base class for the Monte Carlo simulation with the methods `run_Geometric_Brownain_Motion()` and `generate_Paths()`. The `run_Geometric_Brownain_Motion()` method will simulate the underlying asset price paths using the Euler-Maruyama scheme as derived in $\text{Equation } (7)$. The `generate_Paths()` method will call the first method, storing the returned NumPy arrays as attributes in the class object. Later on we will define classes that will inherit this class for the Asian and lookback options, defining the payoff functions for each option type in these child classes.

In [1]:
```
# Install dependencies
import numpy as np
import pandas as pd
import scipy.stats as ss
import tabulate
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from tqdm import tqdm
import time
from typing import Tuple

# Configure matplotlib plots for A4 paper
plt.rcParams['figure.figsize'] = (11, 8)
plt.rcParams['axes.titlesize'] = 12
plt.rcParams['figure.dpi'] = 200
plt.rcParams['mathtext.fontset'] = 'cm'
```

```
plt.rcParams['grid.color'] = '#F0F0F0'
plt.rcParams['grid.linestyle'] = 'solid'

# Define constants
TRADING_DAYS_PER_YEAR = 252
NUMBER_OF_SIMULATIONS = 10000
```

In [2]:

```
class MonteCarloSimulation:
    ''' Class for a Monte Carlo simulation using the Euler-Maruyama scheme to
    simulate the underlying asset price paths.
    Args:
        S0 (float): The initial underlying asset price
        E (float): The strike price
        T (float): The time horizon or time to maturity in years
        r (float): Risk-free rate as the drift coefficient
        sigma (float): Volatility
        number_of_paths (int): The number of path realisations
        random_seed (int): The random seed for reproducibility [default: 0]

    Returns:
        None
    '''
    def __init__(self, S0: float, E: float, T: float, r: float,
                 sigma: float, number_of_paths: int, random_seed: int):
        self.S0 = S0
        self.E = E
        self.time_Steps = int(T * TRADING_DAYS_PER_YEAR)
        self.T = T
        self.r = r
        self.sigma = sigma
        self.number_Of_Paths = number_of_paths
        self.random_Seed = random_seed

    def run_GBM(self) -> Tuple[np.ndarray, np.ndarray]:
        ''' Simulate the underlying asset price paths assuming they follow a
        risk-neutral Geoemtric Brownian Motion process using the
        Euler-Maruyama scheme.
        Args:
            None

        Returns:
            St (np.ndarray): An array of the simulated asset price paths with
            shape (time_steps, number_of_paths)
            time_axis (np.ndarray): An array of the time axis used for plotting
            with shape (time_steps, )
        '''
        # Set the random seed
        np.random.seed(self.random_Seed)

        # Calculate dt and time axis used for plotting
        T = self.T

        time_axis, dt = np.linspace(0, T, self.time_Steps, retstep=True)

        # Set the input paramters, r is the risk-free rate
        r = self.r
```

```
        sigma = self.sigma

        # Simulate paths
        St = np.zeros((self.time_Steps, self.number_Of_Paths))
        St[0] = self.S0

        # Produce an array of standard normal random numbers
        phi = np.random.standard_normal(size=(self.time_Steps,
                                              self.number_Of_Paths))

        # Simulate the GBM process using the Euler-Maruyama scheme
        for t in range(1, self.time_Steps):
            St[t] = St[t-1] * np.exp((r - 0.5 * sigma**2) * dt + sigma
                                     * np.sqrt(dt) * phi[t-1])

        return St, time_axis

    def generate_Paths(self):
        ''' Run the Monte Carlo simulation, storing the paths and time axis arrays
        in the St and tt attributes.'''
        # Time the simulation
        start = time.time()

        # Run the simulation
        self.St, self.time_Axis = self.run_GBM()

        # Calculate the run time
        self.run_Time = time.time() - start
```

2.2 Asian Option Class

We will define a class for the Asian option which will inherit the Monte Carlo simulation class. The methods `call_Price()` and `put_Price()` calculate the price of the fixed strike arithmetic Asian call and put options respectively using the Monte Carlo simulated asset price paths stored in the class object (`self.St`) from the `generate_Paths()` method. The option prices are then the expected present value of a fixed strike arithmetic Asian call, C, and put, P, option with time to expiry, T given by $\text{Equation } (14)$ and $\text{Equation } (15)$ respectively.

In [3]:
```
class AsianOption(MonteCarloSimulation):
    def __init__(self, S0: float, E: float, T: float, r: float,
                 sigma: float, number_of_paths: int, random_seed):
        super().__init__(S0, E, T, r, sigma, number_of_paths, random_seed)

    def call_Price(self, standard_error: bool = False) -> float:
        ''' Calculate the price of a fixed strike arithmetic Asian call option
        with time to maturity T by discounting the average payoff
        Args:
            standard_error (bool): Whether to calculate the standard error of the
            call option price

        Returns:
            C (float): The price of the call option under the risk-neutral measure
```

```
        '''
        # Parameters for readability
        St = self.St
        E = self.E
        T = self.T
        r = self.r

        # Calculate the arithmetic average (A) of the underlying asset prices
        A = np.mean(St, axis=0)

        # Calculate the payoff of each path
        CT = np.maximum(A - E, 0)

        # Calculate the price of the call option
        C = np.exp(-r * T) * np.mean(CT)

        if standard_error:
            # Calculate the standard error of the call option price
            SE = np.std(CT) / np.sqrt(self.number_Of_Paths)
            return C, SE

        else:
            return C

    def put_Price(self, standard_error: bool = False) -> float:
        ''' Calculate the price of a fixed strike arithmetic Asian put option with
        time to maturity T by discounting the average payoff
        Args:
            standard_error (bool): Whether to calculate the standard error of the
            call option price

        Returns:
            C (float): The price of the put option under the risk-neutral measure
        '''
        # Parameters for readability
        St = self.St
        E = self.E
        T = self.T
        r = self.r

        # Calculate the arithmetic average (A) of the underlying asset prices
        A = np.mean(St, axis=0)

        # Calculate the payoff of each path
        PT = np.maximum(E - A, 0)

        # Calculate the price of the put option
        P = np.exp(-r * T) * np.mean(PT)

        if standard_error:
            # Calculate the standard error of the put option price
            SE = np.std(PT) / np.sqrt(self.number_Of_Paths)
            return P, SE

        else:
            return P
```

2.3 Lookback Option Classes

We will define classes for the maximum and minimum lookback options which will inherit the Monte Carlo simulation class. The methods `call_Price()` and `put_Price()` calculate the price of the fixed strike maximum or minimum lookback call and put options respectively using the Monte Carlo simulated asset price paths stored in the class object (`self.St`) from the `generate_Paths()` method.

The option prices are then the expected present value of a fixed strike maximum or minimum lookback call, C, and put, P, option with time to expiry, T given by $\text{Equation } (22)$ and $\text{Equation } (23)$ respectively.

In [4]:
```
class MaximumLookbackOption(MonteCarloSimulation):
    def __init__(self, S0: float, E: float, T: float, r: float,
                 sigma: float, number_of_paths: int, random_seed):
        super().__init__(S0, E, T, r, sigma, number_of_paths, random_seed)

    def call_Price(self, standard_error: bool = False) -> float:
        ''' Calculate the price of a fixed strike lookback call option with
        time to maturity T by discounting the discretely sampled maximum payoff
        Args:
            standard_error (bool): Whether to calculate the standard error of the
            call option price

        Returns:
            C (float): The price of the call option under the risk-neutral measure
        '''
        # Parameters for readability
        St = self.St
        E = self.E
        T = self.T
        r = self.r

        # Calculate the maximum (M) of the underlying asset prices
        M = np.max(St, axis=0)

        # Calculate the payoff of each path
        CT = np.maximum(M - E, 0)

        # Calculate the price of the call option
        C = np.exp(-r * T) * np.mean(CT)

        if standard_error:
            # Calculate the standard error of the call option price
            SE = np.std(CT) / np.sqrt(self.number_Of_Paths)
            return C, SE

        else:
            return C

    def put_Price(self, standard_error: bool = False) -> float:
        ''' Calculate the price of a fixed strike lookback put option with
```

```
        time to maturity T by discounting the discretely sampled maximum payoff
        Args:
            standard_error (bool): Whether to calculate the standard error of the
            call option price

        Returns:
            P (float): The price of the put option under the risk-neutral measure
        '''
        # Parameters for readability
        St = self.St
        E = self.E
        T = self.T
        r = self.r

        # Calculate the maximum (M) of the underlying asset prices
        M = np.max(St, axis=0)

        # Calculate the payoff of each path
        PT = np.maximum(E - M, 0)

        # Calculate the price of the put option
        P = np.exp(-r * T) * np.mean(PT)

        if standard_error:
            # Calculate the standard error of the put option price
            SE = np.std(PT) / np.sqrt(self.number_Of_Paths)
            return P, SE

        else:
            return P
```

In [5]:

```
class MinimumLookbackOption(MonteCarloSimulation):
    def __init__(self, S0: float, E: float, T: float, r: float,
                 sigma: float, number_of_paths: int, random_seed):
        super().__init__(S0, E, T, r, sigma, number_of_paths, random_seed)

    def call_Price(self, standard_error: bool = False) -> float:
        ''' Calculate the price of a fixed strike minimum lookback call option with
        time to maturity T by discounting the discretely sampled minimum payoff
        Args:
            standard_error (bool): Whether to calculate the standard error of the
            call option price

        Returns:
            C (float): The price of the call option under the risk-neutral measure
        '''
        # Parameters for readability
        St = self.St
        E = self.E
        T = self.T
        r = self.r

        # Calculate the minimum (M) of the underlying asset prices
        M = np.min(St, axis=0)

        # Calculate the payoff of each path
```

```
        CT = np.maximum(M - E, 0)

        # Calculate the price of the call option
        C = np.exp(-r * T) * np.mean(CT)

        if standard_error:
            # Calculate the standard error of the call option price
            SE = np.std(CT) / np.sqrt(self.number_Of_Paths)
            return C, SE

        else:
            return C

    def put_Price(self, standard_error: bool = False) -> float:
        ''' Calculate the price of a fixed strike minimum lookback put option with
        time to maturity T by discounting the discretely sampled minimum payoff
        Args:
            standard_error (bool): Whether to calculate the standard error of the
            call option price

        Returns:
            P (float): The price of the put option under the risk-neutral measure
        '''
        # Parameters for readability
        St = self.St
        E = self.E
        T = self.T
        r = self.r

        # Calculate the minimum (M) of the underlying asset prices
        M = np.min(St, axis=0)

        # Calculate the payoff of each path
        PT = np.maximum(E - M, 0)

        # Calculate the price of the put option
        P = np.exp(-r * T) * np.mean(PT)

        if standard_error:
            # Calculate the standard error of the put option price
            SE = np.std(PT) / np.sqrt(self.number_Of_Paths)
            return P, SE

        else:
            return P
```

2.4 Initial Sample Data Results

First, we will examine fixed strike Asian and lookback options using the sample data below. Then we will vary the data to see the affect on the option prices and the call and put payoffs.

$$\begin{aligned}\text{Today's stock price } S_0 &= 100\\ \text{Strike } E &= 100\\ \text{Time to expiry } (T) &= 1 \text{ year}\\ \text{volatility } \sigma &= 20\%\\ \text{constant risk-free interest rate } r &= 5\%\end{aligned}$$

First we will create a Factory class called `OptionFactory` to initialise an instance of an option depending on the option type specified. This will come in useful when we want to price multiple options using the same parameters.

In [6]:
```python
class OptionFactory:
    ''' Factory class for creating option objects'''

    @staticmethod
    def create_Option(option_type: str, S0: float, E: float, T: float, r: float,
                      sigma: float, random_seed: int):
        ''' Return an option object based on the option type
        Args:
            option_type (str): The type of option to create
            S0 (float): The initial underlying asset price
            E (float): The strike price
            T (float): The time horizon or time to maturity in years
            r (float): Risk-free rate as the drift coefficient
            sigma (float): Volatility
            number_of_paths (int): The number of path realisations
            random_seed (int): The random seed for reproducibility [default: 0]

        Returns:
            option (object): An option object
        '''
        if option_type == 'Asian':
            return AsianOption(S0, E, T, r, sigma,
                               NUMBER_OF_SIMULATIONS, random_seed)
        elif option_type == 'Maximum lookback':
            return MaximumLookbackOption(S0, E, T, r, sigma,
                                         NUMBER_OF_SIMULATIONS, random_seed)
        elif option_type == 'Minimum lookback':
            return MinimumLookbackOption(S0, E, T, r, sigma,
                                         NUMBER_OF_SIMULATIONS, random_seed)
        else:
            raise ValueError('Option type not recognised')
```

In [7]:
```python
# Initialise parameters
S0 = 100
E = 100
T = 1
sigma = 0.2
r = 0.05

# Random seed for reproducibility
random_seed = 0

# Create a list of options
options = ['Asian', 'Maximum lookback', 'Minimum lookback']
```

```
results = []

for option_type in options:
    # Create an option object and generate paths
    option = OptionFactory.create_Option(option_type, S0, E, T, r,
                                         sigma, random_seed)
    option.generate_Paths()

    call = option.call_Price(standard_error=True)
    put = option.put_Price(standard_error=True)
    call_price, call_SE, put_price, put_SE = call[0], call[1], put[0], put[1]
    num_paths, run_time = option.number_Of_Paths, option.run_Time

    results.append([f'{option_type} call', call_price, call_SE,
                    option.number_Of_Paths, option.run_Time])
    results.append([f'{option_type} put', put_price, put_SE,
                    option.number_Of_Paths, option.run_Time])

headers = ['Option', 'Price', 'SE +/-', 'Paths', 'Run Time']
print(tabulate.tabulate(results,
                        headers=headers,
                        floatfmt=('.1f', '.2f', '.3f', '.1f', '.3f'),
                        tablefmt='psql'))
```

```
+----------------------+---------+----------+---------+------------+
| Option               |   Price |   SE +/- |   Paths |   Run Time |
|----------------------+---------+----------+---------+------------|
| Asian call           |    5.76 |    0.083 |   10000 |      0.051 |
| Asian put            |    3.21 |    0.054 |   10000 |      0.051 |
| Maximum lookback call |   18.40 |    0.161 |   10000 |      0.055 |
| Maximum lookback put  |    0.00 |    0.000 |   10000 |      0.055 |
| Minimum lookback call |    0.00 |    0.000 |   10000 |      0.054 |
| Minimum lookback put  |   11.51 |    0.093 |   10000 |      0.054 |
+----------------------+---------+----------+---------+------------+
```

From the table above we can see that as the Asian call and put option prices are comparable. In contrast, the maximum lookback call price is more expensive than the Asian call, and is comparable in price to the minimum lookback put. This is because the maximum lookback call option is more likely to be in the money than the Asian call option, and the minimum lookback put option is more likely to be in the money than the Asian put option due to the nature of the payoff functions. The maximum lookback put and minimum lookback call can never be in the money as the initial asset price, S_0 is equal to the strike price, E. This explains why the maximum lookback put and minimum lookback call option prices are zero.

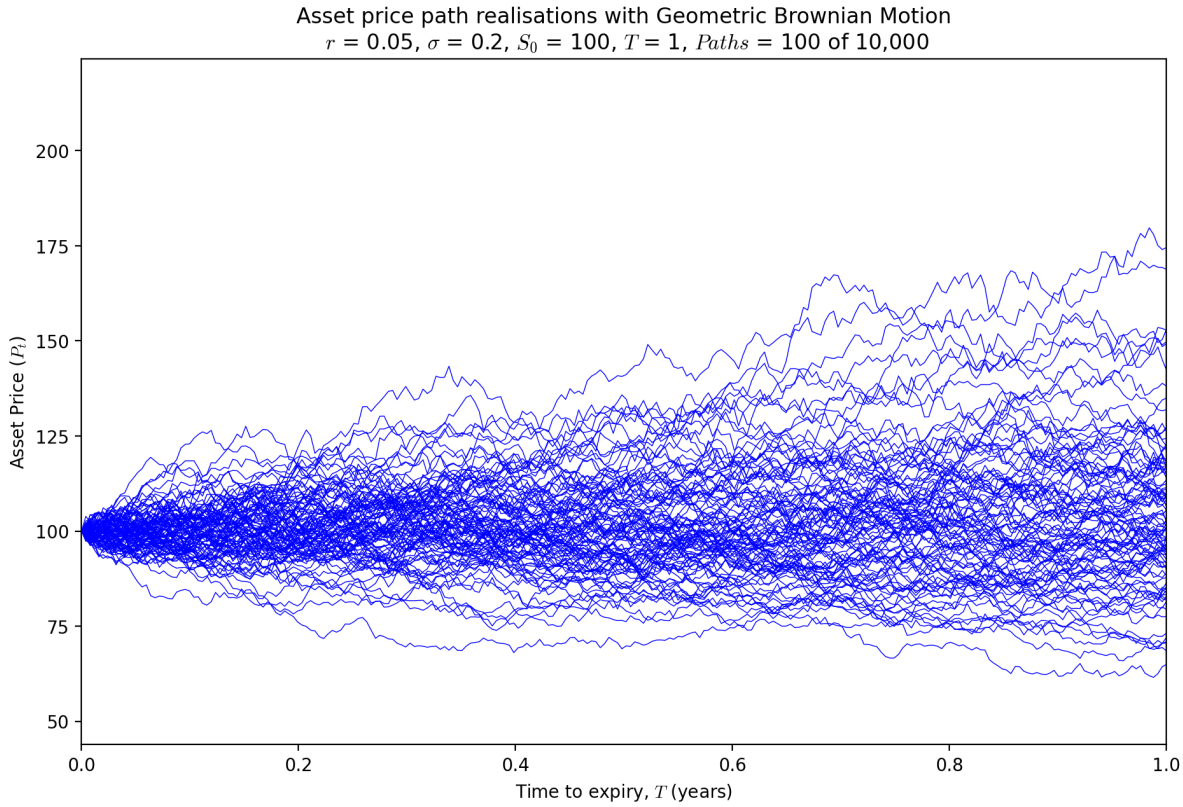
For the option prices who's values are not negative, their standard error's are comparable. The run time of the Monte Carlo simulation is also comparable for all the options as the time step, δt, and the number of paths, N, are the same for all the options.

2.4.1 Underlying Asset Price Paths

We can visualise the simulated asset price paths used by the Asian and lookback options using the Euler-Maruyama scheme. The below plot shows simulated asset price paths with

the intial input data using 10,000 path realisations of the underlying asset price over the time horizon. 100 of the 10,000 simulated asset price paths are plotted for clarity purposes.

In [8]:
```
# Get each of the option objects
asset = MonteCarloSimulation(S0, E, T, r, sigma,
                             NUMBER_OF_SIMULATIONS, random_seed)
asset.generate_Paths()

# Plot 100 randomly selected paths of the 10,000 simulated paths
plt.figure(figsize=(11, 7))

display_paths = 100
for i in np.random.choice(np.array(range(NUMBER_OF_SIMULATIONS)), display_paths):
  plt.plot(asset.time_Axis, asset.St[:, i], color='b', lw=0.5)

plt.xlim(0, T)
plt.ylim(np.min(asset.St), np.max(asset.St))
plt.xlabel('Time to expiry, $T$ (years)')
plt.ylabel('Asset Price ($P_t$)')

title = 'Asset price path realisations with Geometric Brownian Motion' + \
        f'\n $r$ = {r}, $\sigma$ = {sigma}, $S_0$ = {S0}, $T$ = {T}, ' + \
        f'$Paths$ = {display_paths:,d} of {NUMBER_OF_SIMULATIONS:,d}'
plt.title(title)
plt.show()
```

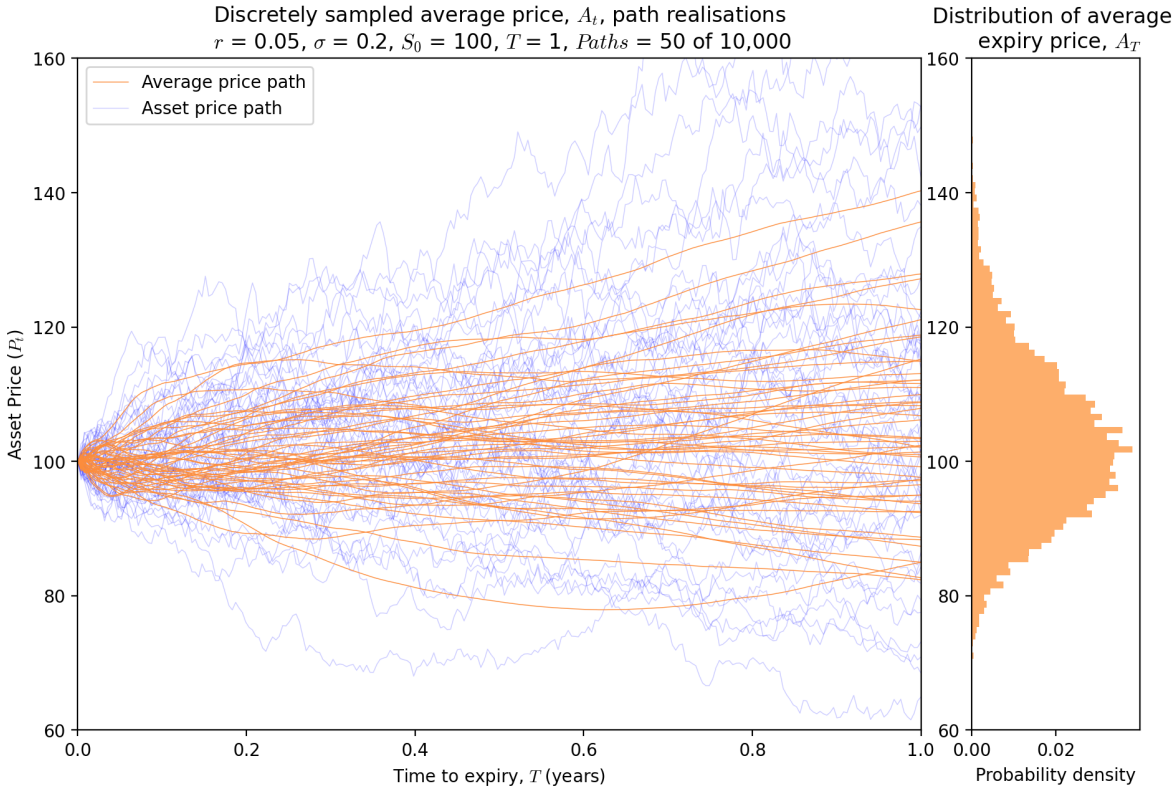


2.4.2 Fixed Strike Asian Option Visualisation

For the Asian option, the discretely sampled arithmetic average of the asset price over the life of the option, $A(0,T)$, is used in place of the underlying asset price in the payoff

function. This is shown in the below plot with the average asset price probability density function (PDF) and histogram plotted alongside the simulated asset price paths.

In [9]:
```
# Set the seed for reproducibility
np.random.seed(2)

# Create an Asian option
asian_option = OptionFactory.create_Option('Asian', S0, E, T, r,
                                           sigma, random_seed)
asian_option.generate_Paths()

# Define the figure and grid
fig = plt.figure(figsize=(11, 7))
grid = gridspec.GridSpec(1, 2, wspace=0.1, hspace=0.3, width_ratios=[5, 1])

# Calculate the running average of the asset prices
discrete_average = np.cumsum(asian_option.St, axis=0) \
                   / np.arange(1, asian_option.time_Steps + 1).reshape(-1, 1)

# Subplot 1: Plot 50 randomly selected asset price paths and their running average
ax1 = plt.subplot(grid[0])
display_paths = 50
for i in np.random.choice(np.array(range(asian_option.number_Of_Paths)),
                          size=display_paths):
    ax1.plot(asian_option.time_Axis, discrete_average[:, i], '#fd8d3c', lw=0.5)
    ax1.plot(asian_option.time_Axis, asian_option.St[:, i], 'b', lw=0.5,
             alpha=0.2, zorder=0)

ax1.set_xlim(0, T)
ax1.set_ylim(60, 160)
ax1.set_xlabel('Time to expiry, $T$ (years)')
ax1.set_ylabel('Asset Price ($P_t$)')
title = 'Discretely sampled average price, $A_t$, path realisations' + \
        f'\n $r$ = {r}, $\sigma$ = {sigma}, $S_0$ = {S0}, $T$ = {T}, ' + \
        f'$Paths$ = {display_paths:,d} of {NUMBER_OF_SIMULATIONS:,d}'
ax1.set_title(title)
ax1.legend(['Average price path', 'Asset price path'], loc='upper left')

# Subplot 2: Plot the histogram of the average asset expiry price for each path
ax2 = plt.subplot(grid[1])
A = discrete_average[-1, :]

# Plot the histogram and pdf
ax2.hist(A, bins=100, density=True, histtype='stepfilled',
         orientation='horizontal', color='#fdae6b', alpha=1)

ax2.set_ylim(60, 160)
ax2.set_xlabel('Probability density')
ax2.set_title('Distribution of average \n expiry price, $A_T$')

# Show the plots
plt.show()
```

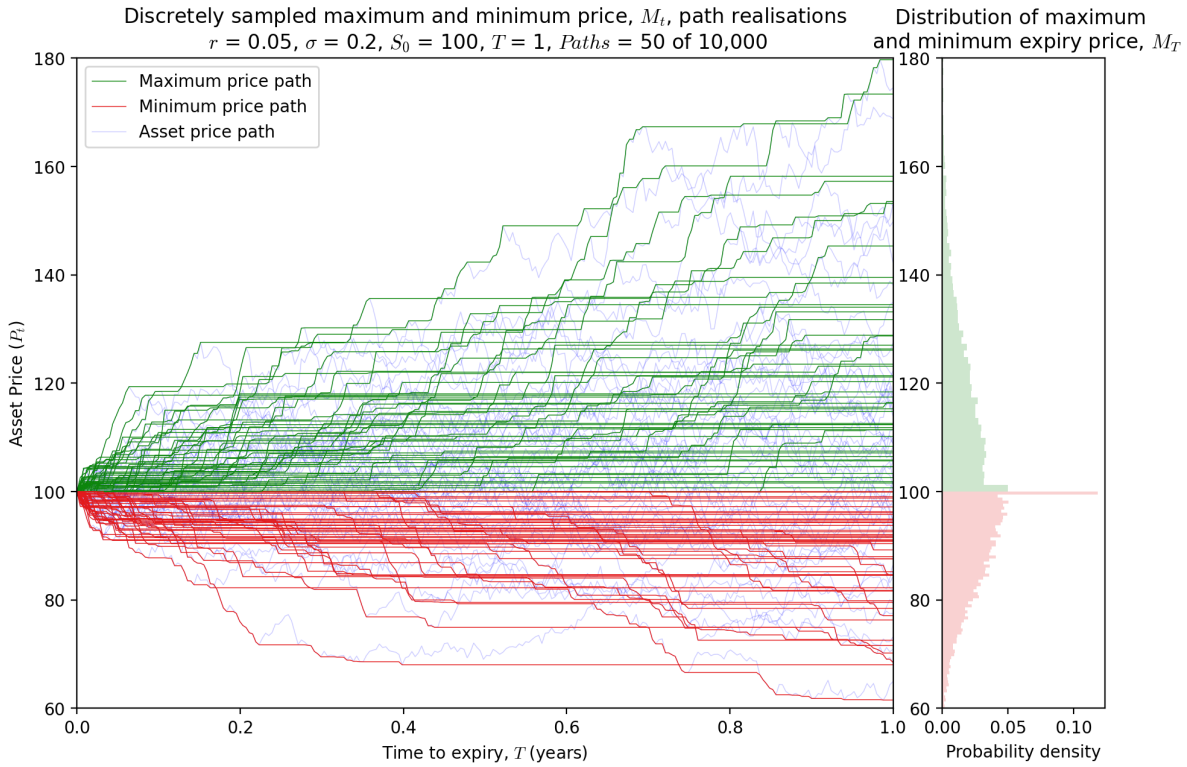


2.4.2 Fixed Strike Maximum and Minimum Lookback Option Visualisation

For the lookback option, the discretely sampled maximum or minimum of the asset price over the life of the option, $M(0,T)$, is used in place of the underlying asset price in the payoff function. This is shown in the below plot with the maximum price probability density function (PDF) and histogram plotted alongside the simulated asset price paths.

In [10]:
```
# Set the seed for reproducibility
np.random.seed(3)

# Create a maximum and minimum lookback option
max_lookback_option = OptionFactory.create_Option('Maximum lookback', S0, E, T, r,
                                                  sigma, random_seed)
min_lookback_option = OptionFactory.create_Option('Minimum lookback', S0, E, T, r,
                                                  sigma, random_seed)
max_lookback_option.generate_Paths()
min_lookback_option.generate_Paths()

# Define the figure and grid
fig = plt.figure(figsize=(11, 7))
grid = gridspec.GridSpec(1, 2, wspace=0.1, hspace=0.3, width_ratios=[5, 1])

# Calculate the running maximum of the asset prices
discrete_maximum = np.maximum.accumulate(max_lookback_option.St, axis=0)
discrete_minimum = np.minimum.accumulate(min_lookback_option.St, axis=0)

# Subplot 1: Plot 50 randomly selected asset price paths and their running average
ax1 = plt.subplot(grid[0])
display_paths = 50
for i in np.random.choice(np.array(range(max_lookback_option.number_Of_Paths)),
```

```
                          size=display_paths):
    ax1.plot(max_lookback_option.time_Axis, discrete_maximum[:, i],
             '#0C860C', lw=0.5)
    ax1.plot(min_lookback_option.time_Axis, discrete_minimum[:, i],
             '#E41A1C', lw=0.5)
    ax1.plot(max_lookback_option.time_Axis, max_lookback_option.St[:, i],
             'b', lw=0.5, alpha=0.2, zorder=0)

ax1.set_xlim(0, T)
ax1.set_ylim(60, 180)
ax1.set_xlabel('Time to expiry, $T$ (years)')
ax1.set_ylabel('Asset Price ($P_t$)')
title = 'Discretely sampled maximum and minimum price, $M_t$, path realisations' +
        f'\n $r$ = {r}, $\sigma$ = {sigma}, $S_0$ = {S0}, $T$ = {T}, ' + \
        f'$Paths$ = {display_paths:,d} of {NUMBER_OF_SIMULATIONS:,d}'
ax1.set_title(title)
ax1.legend(['Maximum price path', 'Minimum price path', 'Asset price path'],
           loc='upper left')

# Subplot 2: Plot the histogram of the maximum asset price for each path
ax2 = plt.subplot(grid[1])
M = discrete_maximum[-1, :]
m = discrete_minimum[-1, :]

# Plot the histogram
ax2.hist(M, bins=100, density=True, histtype='stepfilled',
         orientation='horizontal', color='#0C860C', alpha=0.2)
ax2.hist(m, bins=100, density=True, histtype='stepfilled',
         orientation='horizontal', color='#E41A1C', alpha=0.2)

ax2.set_ylim(60, 180)
ax2.set_xlabel('Probability density')
ax2.set_title('Distribution of maximum \n and minimum expiry price, $M_T$',
              fontsize=12)

# Show the plots
plt.show()
```



2.5 Varying the Input Data

2.5.1 Varying the Initial Asset Price and Time to Expiry

We will now vary the input data to see the affect on the option prices. We will vary time to expiry, T and the initial stock price, S_0, keeping the other input data constant. For each sample of data we will run a Monte Carlo pricing simulation as before, storing the call and put option prices for the Asian and lookback option in their own respective NumPy arrays. Then we will plot the call and put option prices against the initial asset price, S_0, and the time to expiry, T.

In [11]:
```
# Get static parameters
E = 100
sigma = 0.2
r = 0.05

# Define number of samples for the time to expiry and asset price
samples = 50

# Generate array of random seeds for each sampling point
random_seeds = np.linspace(0, samples**2, samples**2, dtype=int)

# Generate arrays containing the range of input values
asset_prices = np.linspace(0, 200, samples)
time_to_expiries = np.linspace(0.01, 1, samples)

# Initialise arrays to store option prices
asian_call_prices = np.zeros((len(asset_prices), len(time_to_expiries)))
asian_put_prices = np.zeros((len(asset_prices), len(time_to_expiries)))
```

```
max_lookback_call_prices = np.zeros((len(asset_prices), len(time_to_expiries)))
max_lookback_put_prices = np.zeros((len(asset_prices), len(time_to_expiries)))

min_lookback_call_prices = np.zeros((len(asset_prices), len(time_to_expiries)))
min_lookback_put_prices = np.zeros((len(asset_prices), len(time_to_expiries)))

# Define the progress bar and monte carlo count for tracking
progress_bar = tqdm(asset_prices, desc='Asset Price', leave=False,
                    total=(len(asset_prices) * len(time_to_expiries)))
monte_carlo_count = 0

# Loop through each asset price and time to expiry
for i, S0 in enumerate(progress_bar):
    for j, T in enumerate(time_to_expiries):
        description = f'Asset Price: {S0:.2f}, Time to Expiry: {T:.2f}'
        progress_bar.set_description(description)

        # Initialise an Asian, maximum lookback and minimum lookback option object
        options = ['Asian', 'Maximum lookback', 'Minimum lookback']
        options = [OptionFactory.create_Option(option,
                                               S0, E, T, r,
                                               sigma, random_seed)
                                               for option in options]

        # Generate paths for each option
        for option in options:
            option.generate_Paths()
            monte_carlo_count += 1

        # Calculate call option prices
        asian_call_prices[i, j], \
        max_lookback_call_prices[i, j], \
        min_lookback_call_prices[i, j] = [option.call_Price() for option in options

        # Calculate put option prices
        asian_put_prices[i, j], \
        max_lookback_put_prices[i, j], \
        min_lookback_put_prices[i, j] = [option.put_Price() for option in options]

        # Update the progress bar and monte carlo count
        progress_bar.update(1)

print(f'Completed {monte_carlo_count} simulations at {samples**2} sampling points')
```

```
Asset Price: 0.00, Time to Expiry: 0.09:   0%|          | 4/2500 [00:00<00:17, 13
8.73it/s]
```

```
Completed 7500 simulations at 2500 sampling points
```

Visualising the Option Price Surfaces

We can visualise the relationship between the option price, time to expiry, and asset price by first defining a function `plot_Option_Price_Surface`. This allows us to re-use the

plotting functionality with other input paramters. Using Matplotlib's 3D projection capabilities we can plot the option price surface for the Asian and lookback options.

The below plots shows the relationship between the option price, the initial asset price and time to expiry for the Asian and lookback options. The option price is plotted on the z-axis, the time to expiry is plotted on the x-axis and the asset price is plotted on the y-axis. The option price is calculated using 10,000 path realisations of the underlying asset price over the time horizon.

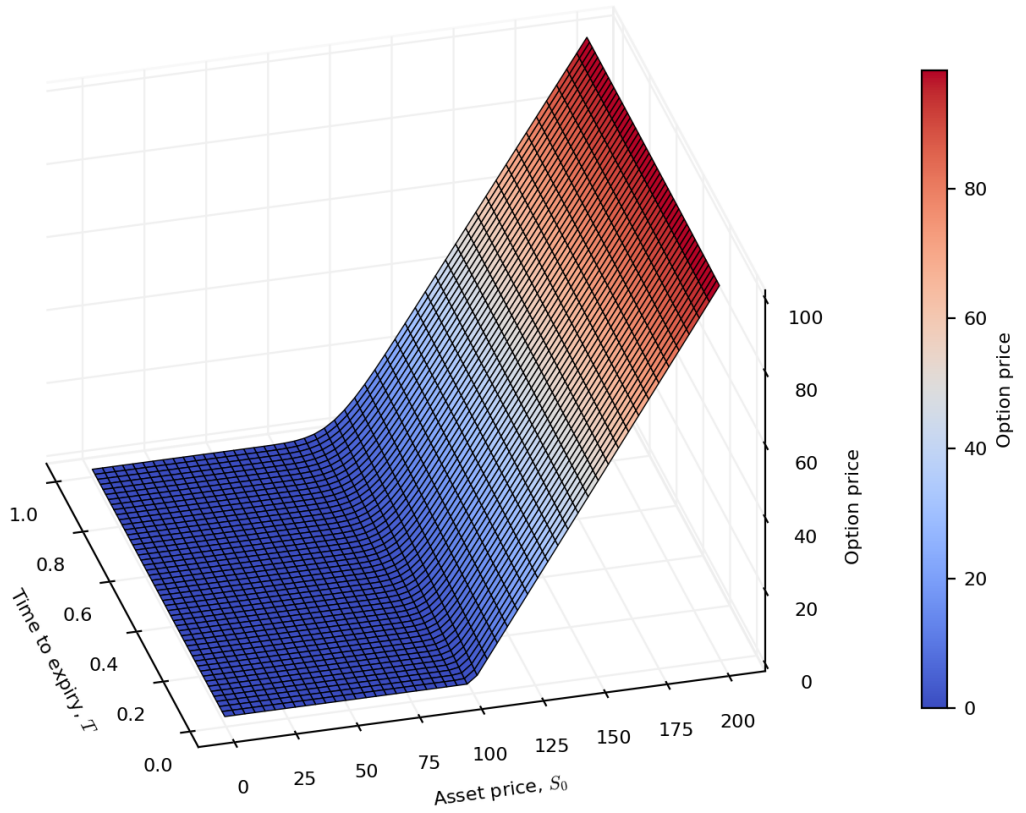
In [12]:
```
def plot_Option_Price_Surface(x: np.ndarray, y: np.ndarray, z: np.ndarray,
                              vertical_rotation: int, horizontal_rotation: int,
                              x_axis_label: str, y_axis_label: str, title: str):
    ''' Plot a 3D surface of the option price as a function of the two parameters
    Args:
        x (np.ndarray): A 1D array of the x-axis values
        y (np.ndarray): A 1D array of the y-axis values
        z (np.ndarray): A 2D array of option prices rows corresponding to x-axis
                        values and columns corresponding to y-axis values
        title (str): The title of the plot

    Returns:
        None
    '''
    x_mesh, y_mesh = np.meshgrid(x, y)

    # Create a figure and grid for the subplots
    fig = plt.figure(figsize=(11, 7))

    # Subplot 1: Plot the 3D surface on the left
    mesh_plot = plt.axes(projection='3d')
    surface = mesh_plot.plot_surface(x_mesh, y_mesh, z, rstride=1, cstride=1,
                                     cmap='coolwarm', shade='interp',
                                     linewidth=0.5, edgecolor='k',
                                     antialiased=True)
    mesh_plot.set_xlabel(x_axis_label, fontsize=8)
    mesh_plot.set_ylabel(y_axis_label, fontsize=8)
    mesh_plot.set_zlabel('Option price', fontsize=8)

    # Plot settings
    mesh_plot.view_init(vertical_rotation, horizontal_rotation)
    mesh_plot.invert_xaxis()
    mesh_plot.set_proj_type('ortho')
    mesh_plot.xaxis.pane.fill = False
    mesh_plot.yaxis.pane.fill = False
    mesh_plot.zaxis.pane.fill = False
    mesh_plot.xaxis.set_tick_params(labelsize=8)
    mesh_plot.yaxis.set_tick_params(labelsize=8)
    mesh_plot.zaxis.set_tick_params(labelsize=8)
    plt.title(title, fontsize=10)

    # Subplot 2: Colourbar on the right
    colour_bar = plt.colorbar(surface, ax=mesh_plot, shrink=0.7, aspect=25)
    colour_bar.set_label('Option price', fontsize=8)
    colour_bar.ax.tick_params(labelsize=8)
```

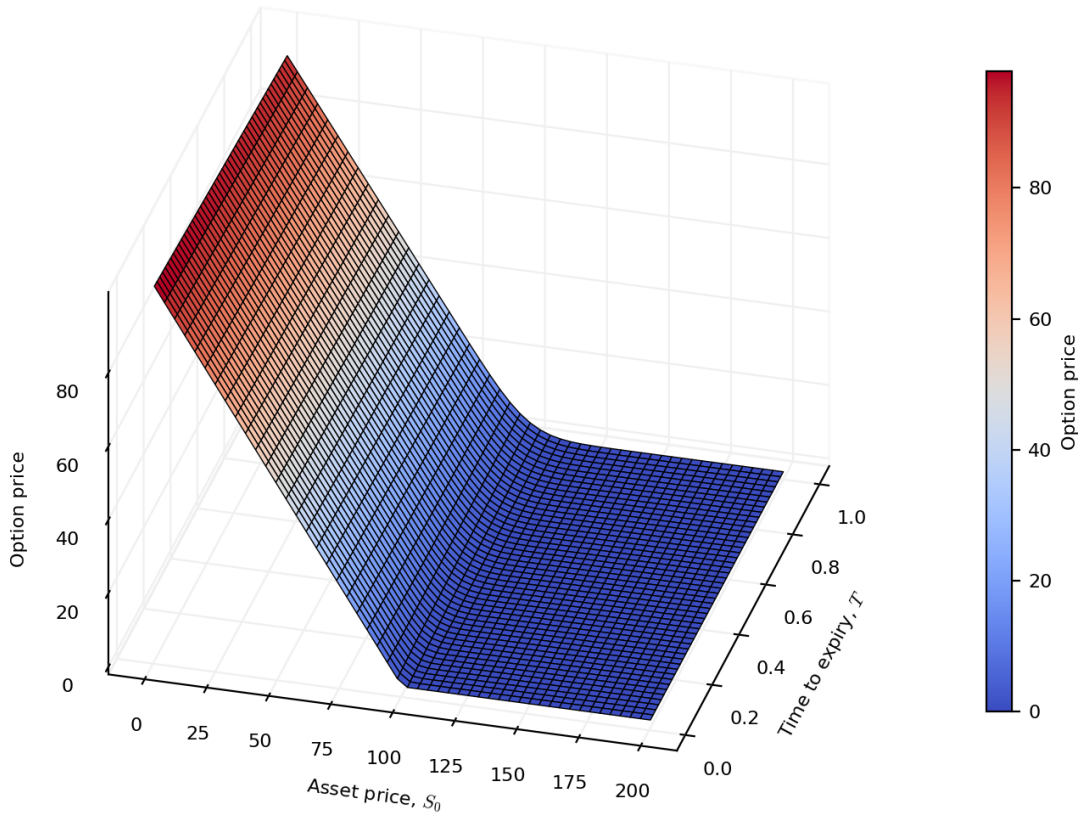
```
    # Show the plot
    plt.show()
```

Fixed Strike Asian Options

In [13]:
```
# Plot the Asian call option price surface
plot_Option_Price_Surface(x=time_to_expiries, y=asset_prices, z=asian_call_prices,
                          vertical_rotation=30, horizontal_rotation=-15,
                          x_axis_label='Time to expiry, $T$',
                          y_axis_label='Asset price, $S_0$',
                          title=f'Fixed Strike Asian Call Option Price, $C$, '
                          'as a function of Asset Price, $S_0$, and '
                          f'Time to Expiry, $T$ \n  $r = {r:.0%}$%, '
                          f'$\sigma = {sigma:.0%}$%, $E = {E}$')

# Plot the Asian put option price surface
plot_Option_Price_Surface(x=time_to_expiries, y=asset_prices, z=asian_put_prices,
                          vertical_rotation=30, horizontal_rotation=15,
                          x_axis_label='Time to expiry, $T$',
                          y_axis_label='Asset price, $S_0$',
                          title=f'Fixed Strike Asian Put Option Price, $P$, '
                          'as a function of Asset Price, $S_0$, and '
                          f'Time to Expiry, $T$ \n  $r = {r:.0%}$%, '
                          f'$\sigma = {sigma:.0%}$%, $E = {E}$')
```

Fixed Strike Asian Call Option Price, C, as a function of Asset Price, S_0, and Time to Expiry, T
$r = 5\%$, $\sigma = 20\%$, $E = 100$





For the fixed strike Asian call and put options there is not much variation in the option price with respect to time to expiry, this is because the average asset price is calculated over the life of the option. However, there is a large variation in the option price with respect to the initial asset price. This is because the average asset price is calculated over the life of the option and therefore with a fixed volatility $\sigma = 20\%$, and strike price $E = 100$, the option price is much more sensitive to the varying initial asset price S_0.

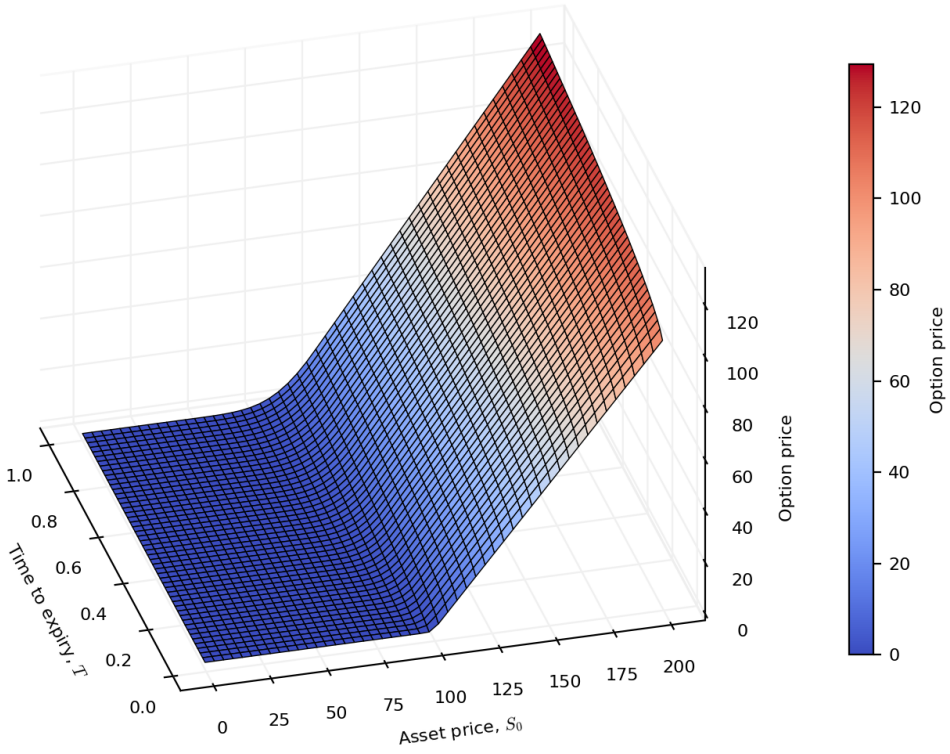
The call and put option prices have a similar range, with values between 0 and 101.

Fixed Strike Maximum Lookback Options

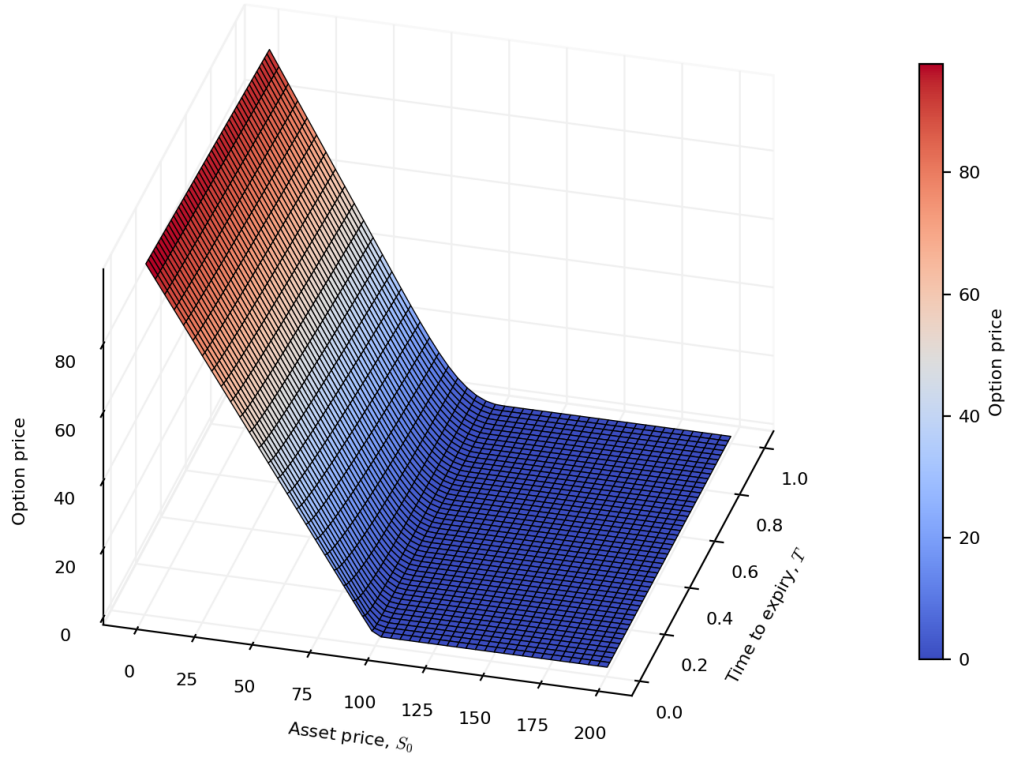
In [14]:
```
# Plot the maximum lookback call option price surface
plot_Option_Price_Surface(x=time_to_expiries, y=asset_prices, z=max_lookback_call_p
                          vertical_rotation=30, horizontal_rotation=-15,
                          x_axis_label='Time to expiry, $T$',
                          y_axis_label='Asset price, $S_0$',
                          title = f'Fixed Strike Maximum Lookback Call Option Price
                          '$C$, as a function of Asset Price, $S_0$, and '
                          f'Time to Expiry, $T$ \n  $r = {r:.0%}$%, '
                          f'$\sigma = {sigma:.0%}$%, $E = {E}$')

# Plot the maximum lookback put option price surface
plot_Option_Price_Surface(x=time_to_expiries, y=asset_prices, z=max_lookback_put_pr
```

```
                        vertical_rotation=30, horizontal_rotation=15,
                        x_axis_label='Time to expiry, $T$',
                        y_axis_label='Asset price, $S_0$',
                        title=f'Fixed Strike Maximum Lookback Put Option Price, '
                        '$P$, as a function of Asset Price, $S_0$, and '
                        f'Time to Expiry, $T$ \n  $r = {r:.0%}$%, '
                        f'$\sigma = {sigma:.0%}$%, $E = {E}$')
```

Fixed Strike Maximum Lookback Call Option Price, C, as a function of Asset Price, S_0, and Time to Expiry, T
$r = 5$%, $\sigma = 20$%, $E = 100$



Fixed Strike Maximum Lookback Put Option Price, P, as a function of Asset Price, S_0, and Time to Expiry, T
$r = 5\%$, $\sigma = 20\%$, $E = 100$



The fixed strike maximum lookback put option plot looks very similar to the fixed strike Asian put option plot. The variation of option price with respect to time to expiry is very small as before, however there is a large variation in the option price with respect to the initial asset price S_0. They appear similar because both options are in the money when either the maximum price or average price up until expiry is less than the strike price. Therefore, we see more of an effect on the option price as we vary the initial asset price S_0 rather than time to expiry T.

In contrast, the fixed strike maximum lookback call is in the money when the maximum price up until expiry is above the strike price, E. This causes the the option price to be even more sensitive to the time to expiry T rather than the initial asset price S_0 once the srike price has been exceeded as the payoff can only increase with time to expiry.

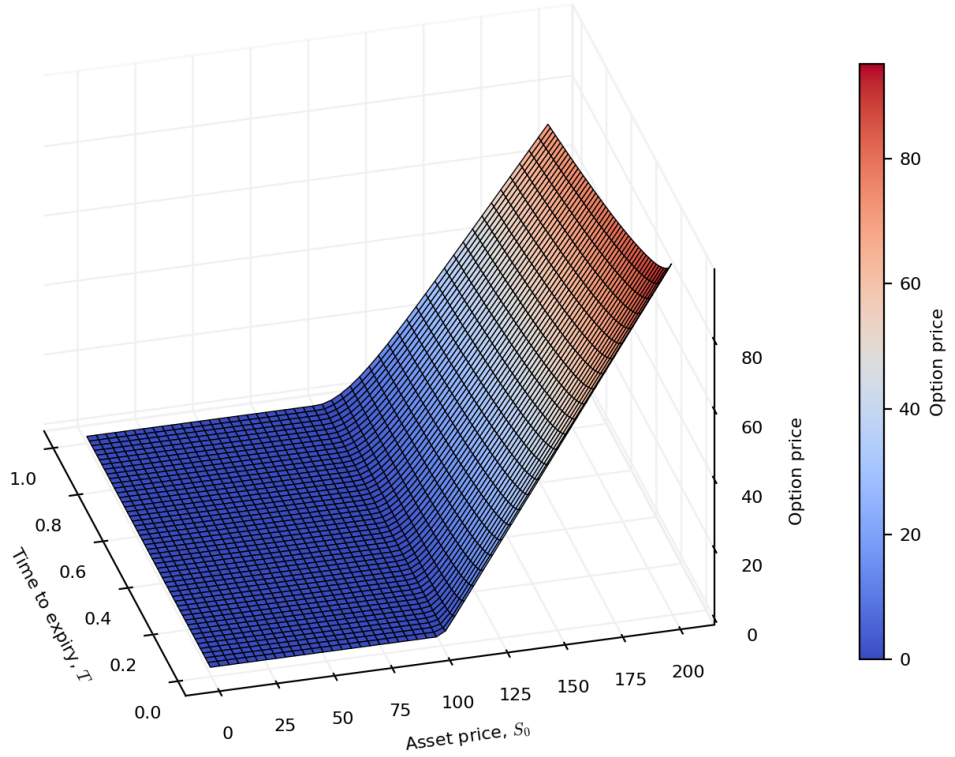
The range of values for the call option is therefore larger (0 to 132) due to the potentially uncapped upside of the option price. The range of values for the put option is similar to the Asian put option (0 to 100).

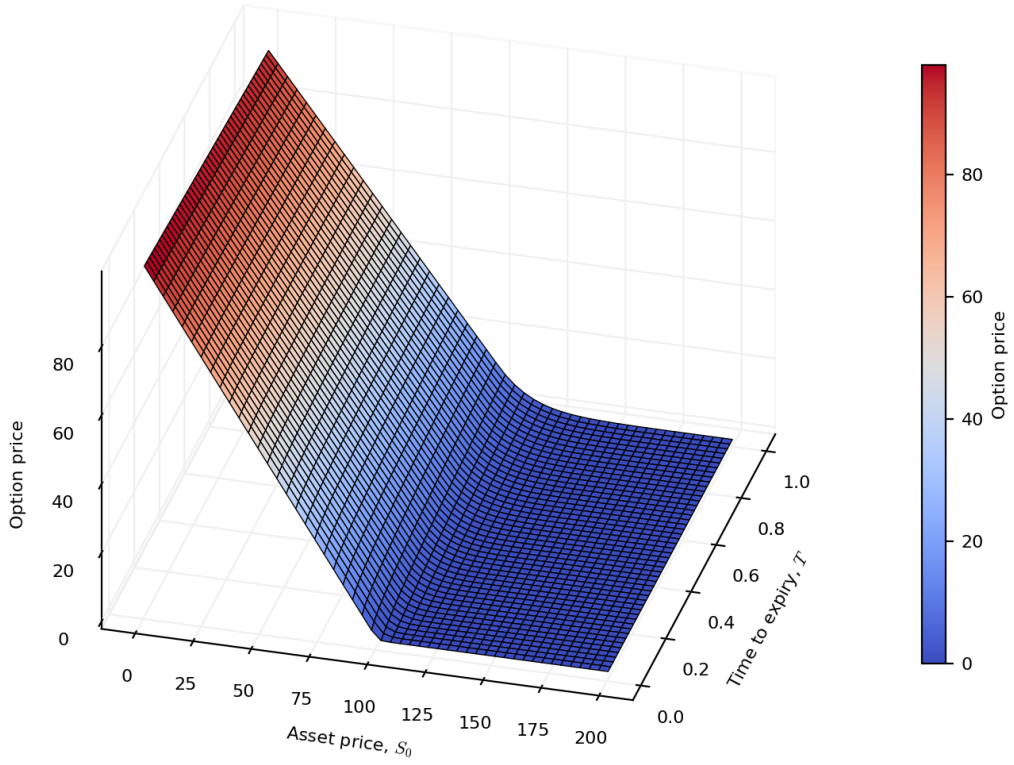
Fixed Strike Minimum Lookback Options

In [15]:
```
# Plot the minimum lookback call option price surface
plot_Option_Price_Surface(x=time_to_expiries, y=asset_prices, z=min_lookback_call_p
                          vertical_rotation=30, horizontal_rotation=-15,
                          x_axis_label='Time to expiry, $T$',
```

```
                          y_axis_label='Asset price, $S_0$',
                          title = f'Fixed Strike Minimum Lookback Call Option Price
                          '$C$, as a function of Asset Price, $S_0$, and '
                          f'Time to Expiry, $T$ \n  $r = {r:.0%}$%, '
                          f'$\sigma = {sigma:.0%}$%, $E = {E}$')

# Plot the minimum lookback put option price surface
plot_Option_Price_Surface(x=time_to_expiries, y=asset_prices, z=min_lookback_put_pr
                          vertical_rotation=30, horizontal_rotation=15,
                          x_axis_label='Time to expiry, $T$',
                          y_axis_label='Asset price, $S_0$',
                          title=f'Fixed Strike Minimum Lookback Put Option Price, '
                          '$P$, as a function of Asset Price, $S_0$, and '
                          f'Time to Expiry, $T$ \n  $r = {r:.0%}$%, '
                          f'$\sigma = {sigma:.0%}$%, $E = {E}$')
```

Fixed Strike Minimum Lookback Call Option Price, C, as a function of Asset Price, S_0, and Time to Expiry, T
$r = 5$%, $\sigma = 20$%, $E = 100$





For the fixed stike minimum lookback option, the put option price surface looks very similar to the previous Asian and maximum lookback put option surfaces. This is because the put option is in the money when the minimum price up until expiry is less than the strike price, E and minimum price can only decrease below the the strike price with time to expiry. For the call option surface, the option has the highest value when time to expiry, T is very small and the initial asset price, S_0 is above the strike price, E. This also makes sense, because as time to expiry increases the underlying asset price has more time to decrease below the strike price, E and this would be taken as the minimum in the payoff function.

2.5.2 Varying the Initial Asset Price and Volatility

We will now vary volatility, σ and the initial stock price, S_0, keeping the other input data constant. For each sample of data we will run a Monte Carlo pricing simulation as before, then plot the call and put option prices against the initial asset price, S_0, and the volatility, σ.

In [16]:
```
# Get static parameters
E = 100
T = 1
r = 0.05

# Define number of samples for the time to expiry and asset price
samples = 50
```

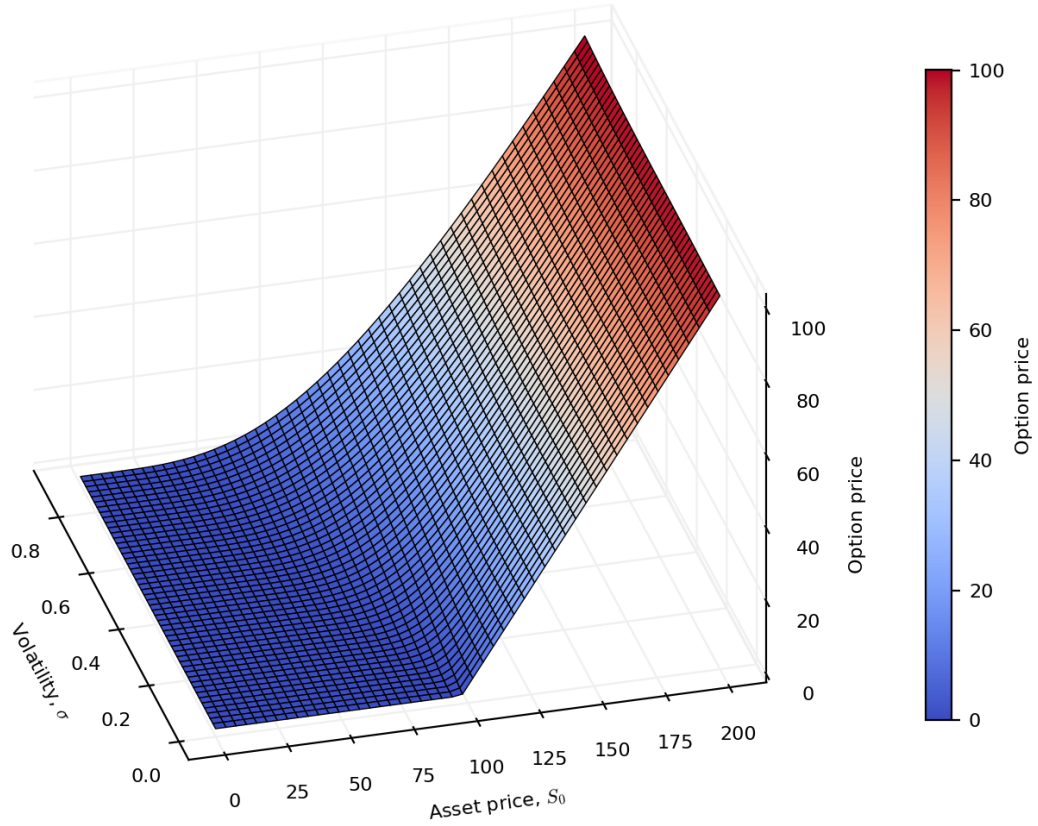
```python
# Generate array of random seeds for each sampling point
random_seeds = np.linspace(0, samples**2, samples**2, dtype=int)

# Generate arrays containing the range of input values
asset_prices = np.linspace(0, 200, samples)
volatilities = np.linspace(0.001, 0.9, samples)

# Initialise arrays to store option prices
asian_call_prices = np.zeros((len(asset_prices), len(volatilities)))
asian_put_prices = np.zeros((len(asset_prices), len(volatilities)))

max_lookback_call_prices = np.zeros((len(asset_prices), len(volatilities)))
max_lookback_put_prices = np.zeros((len(asset_prices), len(volatilities)))

min_lookback_call_prices = np.zeros((len(asset_prices), len(volatilities)))
min_lookback_put_prices = np.zeros((len(asset_prices), len(volatilities)))

# Define the progress bar and monte carlo count for tracking
progress_bar = tqdm(asset_prices, leave=False,
                    total=(len(asset_prices) * len(volatilities)))
monte_carlo_count = 0

# Loop through each asset price and time to expiry
for i, S0 in enumerate(progress_bar):
    for j, sigma in enumerate(volatilities):
        description = f'Asset Price: {S0:.2f}, Volatility: {sigma:.2f}'
        progress_bar.set_description(description)

        # Initialise an Asian, maximum lookback and minimum lookback option object
        options = ['Asian', 'Maximum lookback', 'Minimum lookback']
        options = [OptionFactory.create_Option(option, S0, E, T, r,
                                               sigma, random_seed)
                                               for option in options]

        # Generate paths for each option
        for option in options:
            option.generate_Paths()
            monte_carlo_count += 1

        # Calculate call option prices
        asian_call_prices[i, j], \
        max_lookback_call_prices[i, j], \
        min_lookback_call_prices[i, j] = [option.call_Price() for option in options

        # Calculate put option prices
        asian_put_prices[i, j], \
        max_lookback_put_prices[i, j], \
        min_lookback_put_prices[i, j] = [option.put_Price() for option in options]

        # Update the progress bar and monte carlo count
        progress_bar.update(1)

print(f'Completed {monte_carlo_count} simulations at {samples**2} sampling points')
```

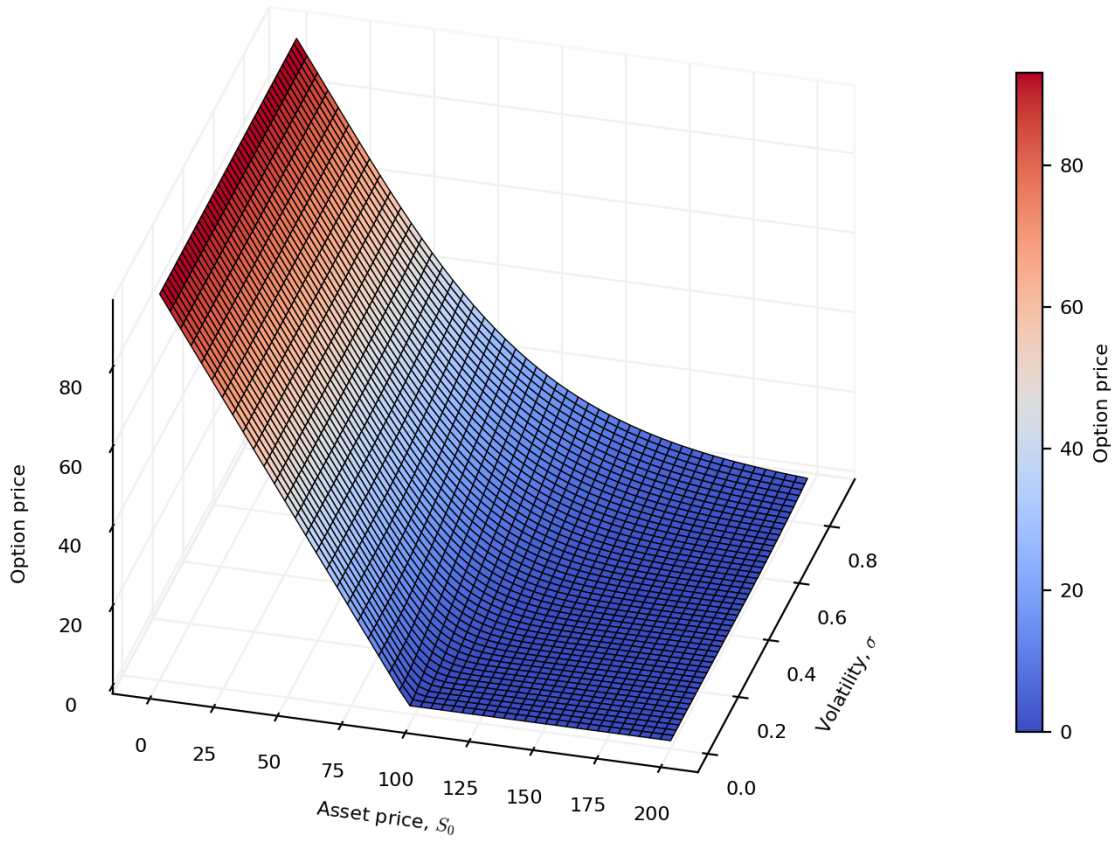
```
Completed 7500 simulations at 2500 sampling points
```

Fixed Strike Asian Options

In [17]:
```
# Plot the Asian call option price surface
plot_Option_Price_Surface(x=volatilities, y=asset_prices, z=asian_call_prices,
                          vertical_rotation=30, horizontal_rotation=-15,
                          x_axis_label='Volatility, $\sigma$',
                          y_axis_label='Asset price, $S_0$',
                          title = f'Fixed Strike Asian Call Option Price, $C$, '
                          'as a function of Asset Price, $S_0$, and Volatility, '
                          f'$\sigma$ \n  $r = {r:.0%}$%, '
                          f'$T = {T}$, $E = {E}$')

# Plot the Asian put option price surface
plot_Option_Price_Surface(x=volatilities, y=asset_prices, z=asian_put_prices,
                          vertical_rotation=30, horizontal_rotation=15,
                          x_axis_label='Volatility, $\sigma$',
                          y_axis_label='Asset price, $S_0$',
                          title = f'Fixed Strike Asian Put Option Price, $P$, '
                          'as a function of Asset Price, $S_0$, and Volatility, '
                          f'$\sigma$ \n  $r = {r:.0%}$%, '
                          f'$T = {T}$, $E = {E}$')
```

Fixed Strike Asian Call Option Price, C, as a function of Asset Price, S_0, and Volatility, σ
$r = 5$%, $T = 1$, $E = 100$





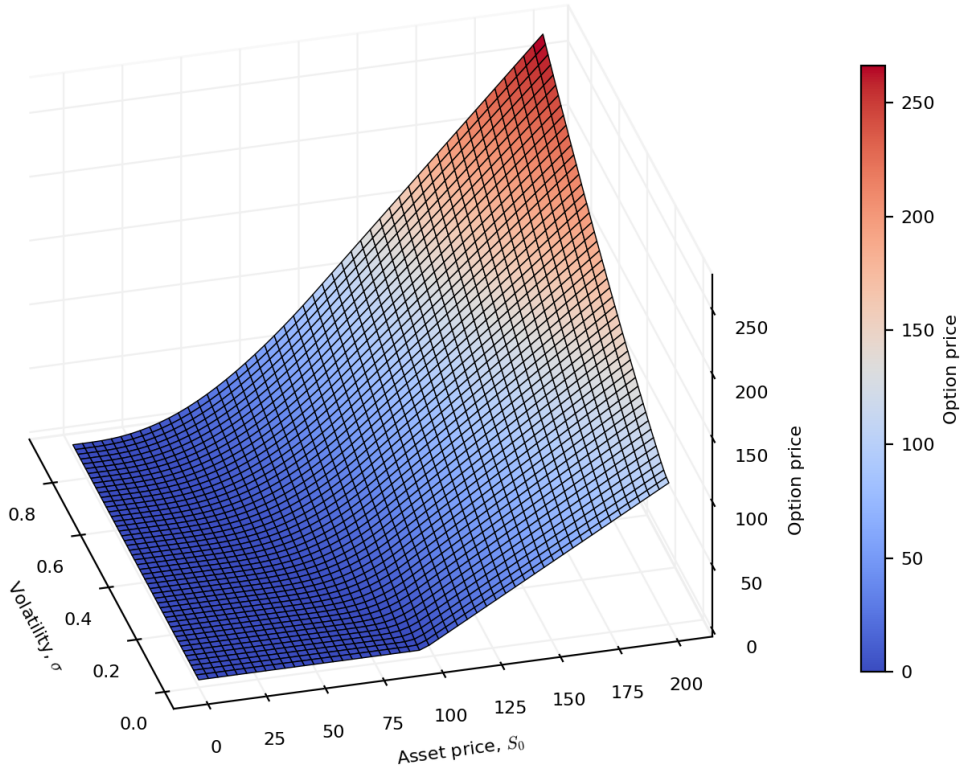
The fixed strike Asian call and put option prices are sensitive to both the initial asset price, S_0, as well as the volatility, σ. For any given initial asset price, S_0, the option price increases with the volatility of the underlying σ. This is due to the fact that the underlying asset price is more likely to move further away from the strike price, E, with a higher volatility and therefore the option has a higher chance of being in the money (or conversely out of the money) at expiry.

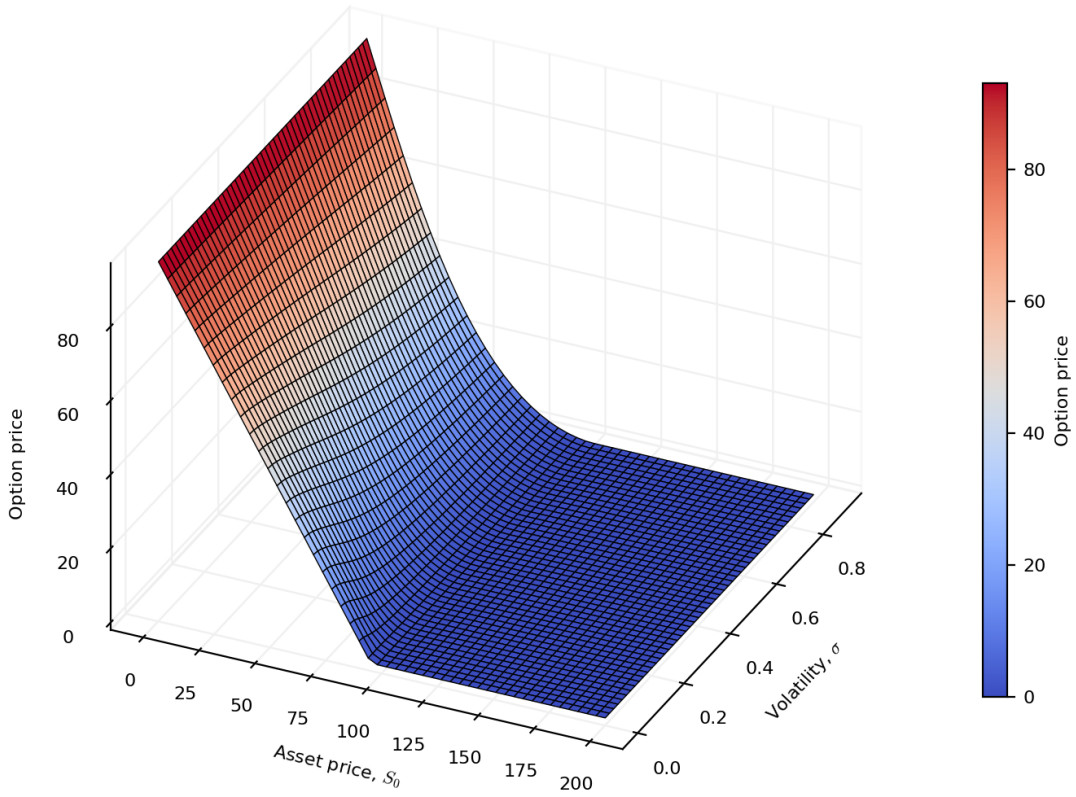
Fixed Strike Maximum Lookback Options

In [18]:
```
# Plot the maximum lookback call option price surface
plot_Option_Price_Surface(x=volatilities, y=asset_prices, z=max_lookback_call_price
                          vertical_rotation=30, horizontal_rotation=-15,
                          x_axis_label='Volatility, $\sigma$',
                          y_axis_label='Asset price, $S_0$',
                          title = f'Fixed Strike Maximum Lookback Call Option Price
                          '$C$, as a function of Asset Price, $S_0$, and Volatility
                          f'$\sigma$ \n  $r = {r:.0%}$%, '
                          f'$T = {T}$, $E = {E}$')

# Plot the maximum lookback put option price surface
plot_Option_Price_Surface(x=volatilities, y=asset_prices, z=max_lookback_put_prices
                          vertical_rotation=30, horizontal_rotation=25,
```

```
                            x_axis_label='Volatility, $\sigma$',
                            y_axis_label='Asset price, $S_0$',
                            title = f'Fixed Strike Maximum Lookback Put Option Price,
                            '$P$, as a function of Asset Price, $S_0$, and Volatility
                            f'$\sigma$ \n  $r = {r:.0%}$%, '
                            f'$T = {T}$, $E = {E}$')
```

Fixed Strike Maximum Lookback Call Option Price, C, as a function of Asset Price, S_0, and Volatility, σ
$r = 5\%$, $T = 1$, $E = 100$





For the maximum lookback call option, when $S_0 > E$, the option is in the money and is more likely to move deeper into the money as volatility increases. Once the strike price has been exceeded, the option payoff is uncapped with increasing volatility as the payoff only considers the maximum historical underlying asset price. This is why the call option price ranges far more widely from 0 to 273, compared to the Asian call option price which ranges from 0 to 132.

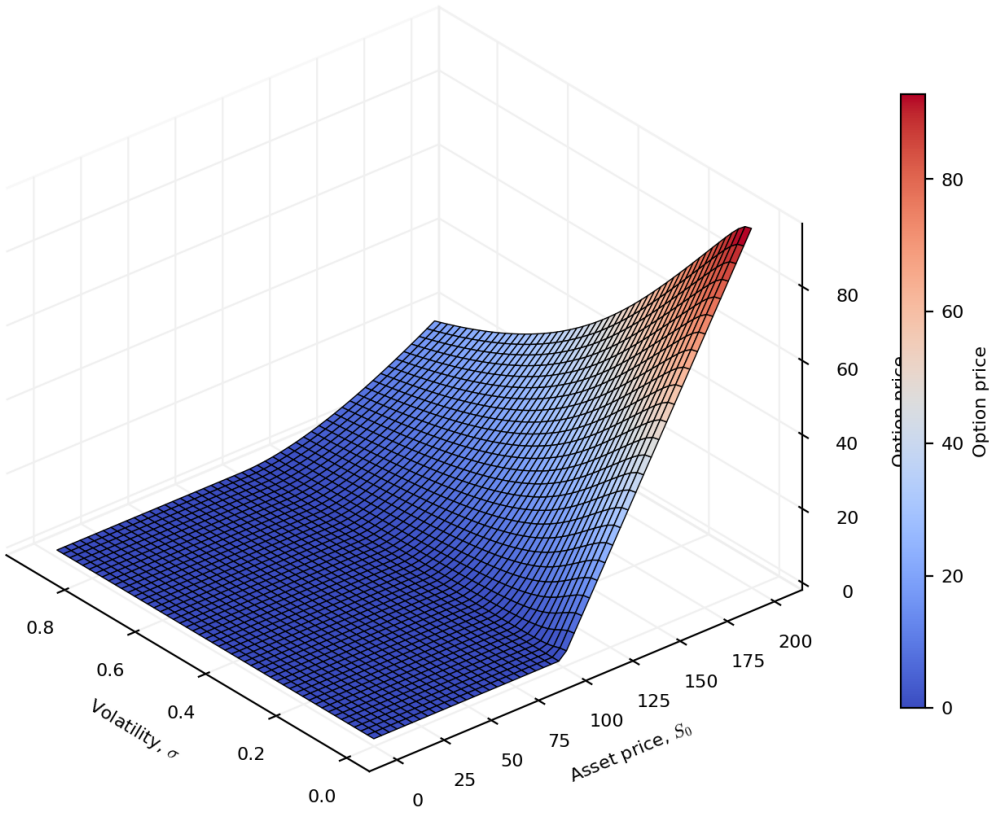
The put option price is more comparable to what we saw when we varied with respect to time to expiry, T and varies between 0 and 96. There is less sensitivity to changing volatility when $S_0 < E$ as the option is out of the money and the payoff is capped by the minimum possible underlying asset price of 0.

Fixed Strike Minimum Lookback Options

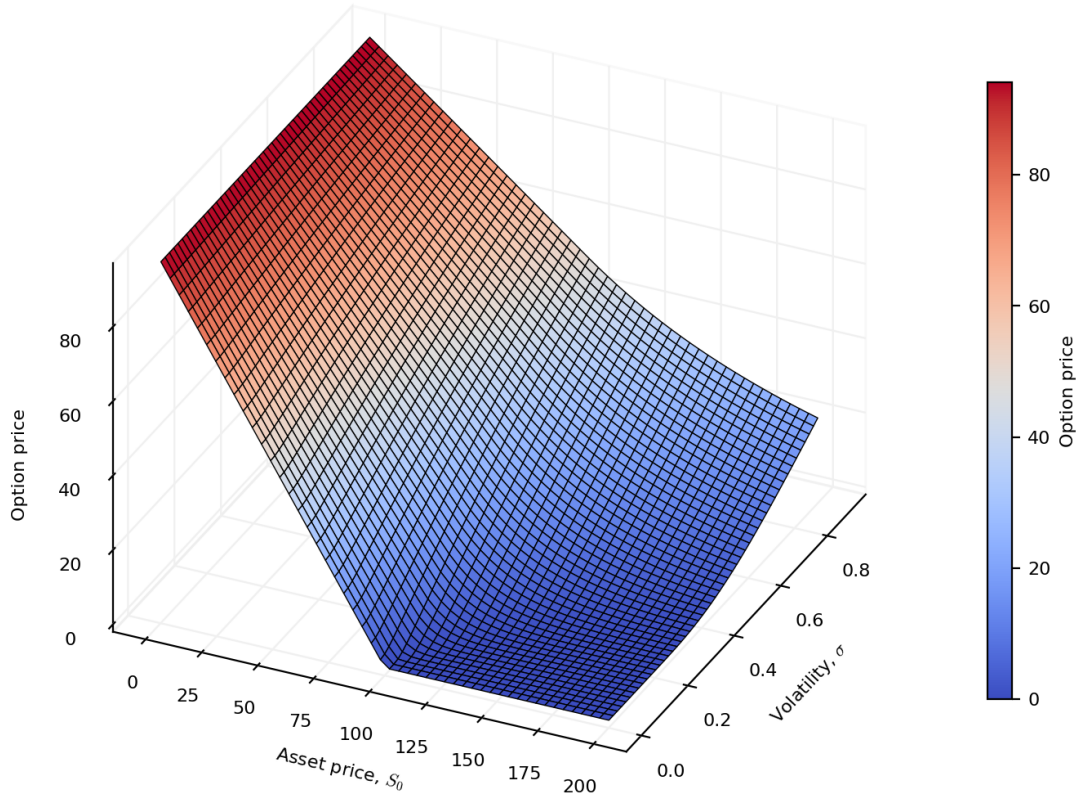
In [19]:
```
# Plot the minimum lookback call option price surface
plot_Option_Price_Surface(x=volatilities, y=asset_prices, z=min_lookback_call_price
                          vertical_rotation=30, horizontal_rotation=-40,
                          x_axis_label='Volatility, $\sigma$',
                          y_axis_label='Asset price, $S_0$',
                          title = f'Fixed Strike Minimum Lookback Call Option Price
                          '$C$, as a function of Asset Price, $S_0$, and Volatility
                          f'$\sigma$ \n  $r = {r:.0%}$%, '
                          f'$T = {T}$, $E = {E}$')
```

```
# Plot the minimum lookback put option price surface
plot_Option_Price_Surface(x=volatilities, y=asset_prices, z=min_lookback_put_prices
                          vertical_rotation=30, horizontal_rotation=25,
                          x_axis_label='Volatility, $\sigma$',
                          y_axis_label='Asset price, $S_0$',
                          title = f'Fixed Strike Minimum Lookback Put Option Price,
                          '$P$, as a function of Asset Price, $S_0$, and Volatility
                          f'$\sigma$ \n  $r = {r:.0%}$%, '
                          f'$T = {T}$, $E = {E}$')
```

Fixed Strike Minimum Lookback Call Option Price, C, as a function of Asset Price, S_0, and Volatility, σ
$r = 5\%$, $T = 1$, $E = 100$



Fixed Strike Minimum Lookback Put Option Price, P, as a function of Asset Price, S_0, and Volatility, σ
$r = 5\%$, $T = 1$, $E = 100$



The fixed strike minimum lookback call option is at its highest when the initial asset price, S_0 is above the strike price, E, and the volatility, σ, is low. This is because even if $S_0 > E$, if the volatility is high then the underlying asset price is more likely to move below the strike price.

For the fixed strike minimum lookback put option, when $S_0 < E$, the option price is higher when volatility is lower. This is because the option is more likely to stay in the money when the volatility is lower. When $S_0 > E$, the option price is higher when volatility is higher. This is because the option is out of the money when $S_0 > E$, but is more likely to become in the money with a higher volatility.

3. Observations and Problems

3.1 Speed and Error of Monte Carlo Simulations

The Monte Carlo technique is a very powerful tool for pricing options, especially in higher dimensions, however it is computationally expensive and due to reliance on generating a finite number of realisations using random numbers, the option price is subject to error and will vary with each run of the simulation.

As discussed in $\text{Section } 1.4$, the error in the option price is governed by the greater of the error due to the discreteness of the time step or the error due to the finite number of paths.

This is given by $\text{Equation } (24)$. The error in the option price is therefore proportional to the square root of the number of paths, N, and inversely proportional to the time step, δt. This means that the error in the option price can be reduced by increasing the number of paths, N, and decreasing the time step, δt. However, this comes at the cost of increased computational time. In other words to improve the accuracy of the option price by a factor of 10 we must perform 100 times as many simulations.

In the follow section we will consider how variance reduction techniques improves the speed and standard error of the Monte Carlo simulations.

3.2 Variance Reduction with Antithetic Variates

We are able to speed up the convergence of the Monte Carlo simulations by using variance reduction techniques as oppose to increasing the number of simulations. The two most common variance reduction techniques are antithetic variates and control variates techniques. We will focus on the antithetic variates technique in this report.

In the antithetic variates technique we calculate two estimates for the option value, one using the original set of Normal random numbers ϕ_i and one using the negative of the original set of Normal random numbers $-\phi_i$. We then take the average of the two estimates to give the final estimate of the option value. This works due to the symmetry of the Normal distribution and works on the same principle as a hedged portfolio having a smaller variance than that of an unhedged portfolio.

Imagining a portfolio consisting of an option written on an asset S_1 and an option written on asset S_2, which is perfectly negatively correlated with S_1 and which has the same price as S_1. Then the risk-neutral random walks of the two assets are given by the SDEs:

$$dS_{1,t} = rS_{1,t}dt + \sigma S_{1,t}dW_t \tag{26}$$

$$dS_{2,t} = rS_{2,t}dt - \sigma S_{2,t}dW_t \tag{27}$$

The value of the two option are identical as the price and volatility of the two assets are the same. However, the variance of the payoff of the portfolio containing both options is considerably smaller than the variance of the payoff of each individual option. This is because the two options are perfectly negatively correlated. When we calculate the average of the option payoffs, the fluctuations in the realisation of one path ($S_{1,t}$) are offset by the fluctuations in its antithetic variate ($S_{2,t}$). The result of this is a smaller spread of estimates around the expected value of the option price.

3.2.1 Antithetic Variates in the Euler-Maruyama Scheme

In the Euler-Maruyama scheme we can use antithetic variates to generate two paths for the underlying asset price, one using ϕ_i and one using $-\phi_i$. We can then use the average of the two paths payoffs to calculate the option price.

The risk-neutral asset price, $S_{1,t}$ and its antithetic variate $S_{2,t}$ using the Euler-Maruyama method over a time step δt are given as:

$$S_{1,t+\delta t} = S_{1,t} \exp\left(\left(r - \frac{1}{2}\sigma^2\right)\delta t + \sigma\sqrt{\delta t}\phi_i\right) \tag{28}$$

$$S_{2,t+\delta t} = S_{2,t} \exp\left(\left(r - \frac{1}{2}\sigma^2\right)\delta t + \sigma\sqrt{\delta t}(-\phi_i)\right) \tag{29}$$

3.2.2 Antithetic Variates with Asian Options

Considering the pricing of fixed strike Asian options with time to expiry, $T - t$, we can write the expected present value of a fixed strike arithmetic Asian call, C and its antithetic variate, $\bar{C}$, option using $\text{Equation } (12)$ and $\text{Equation } (14)$ as:

$$C = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}\left[\max\left(\frac{1}{N}\sum_{i=1}^{N} S_1(t_i) - E, 0\right)\right] \tag{30}$$

$$\bar{C} = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}\left[\max\left(\frac{1}{N}\sum_{i=1}^{N} S_2(t_i) - E, 0\right)\right] \tag{31}$$

Similarly for the fixed strike arithmetic Asian put, P and its antithetic variate, $\bar{P}$, option using $\text{Equation } (12)$ and $\text{Equation } (15)$ as:

$$P = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}\left[\max\left(E - \frac{1}{N}\sum_{i=1}^{N} S_1(t_i), 0\right)\right] \tag{32}$$

$$\bar{P} = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}\left[\max\left(E - \frac{1}{N}\sum_{i=1}^{N} S_2(t_i), 0\right)\right] \tag{33}$$

where $S_1(t_i)$ and $S_2(t_i)$ are the asset prices at time t_i for the original and antithetic variate paths respectively and $\mathbb{E}^{\mathbb{Q}}$ is the expectation under the risk-neutral measure.

3.2.3 Antithetic Variates with Lookback Options

Considering the pricing of fixed strike maximum and minimum lookback options with time to expiry, $T - t$, we can write the expected present value of a fixed strike maximum lookback call, C and its antithetic variate, $\bar{C}$, option using $\text{Equation } (20)$ and $\text{Equation } (22)$ as:

$$C = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}\left[\max\left(\max\left(S_1(t_1), S_1(t_2), \ldots, S_1(t_N)\right) - E, 0\right)\right] \tag{34}$$

$$\bar{C} = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}\left[\max\left(\max\left(S_2(t_1), S_2(t_2), \ldots, S_2(t_N)\right) - E, 0\right)\right] \tag{35}$$

Similarly for the fixed strike maximum lookback put, P and its antithetic variate, $\bar{P}$, option using $\text{Equation } (21)$ and $\text{Equation } (23)$ as:

$$P = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}\left[\max\left(E - \max\left(S_1(t_1), S_1(t_2), \ldots, S_1(t_N)\right), 0\right)\right] \tag{36}$$

$$\bar{P} = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}\left[\max\left(E - \max\left(S_2(t_1), S_2(t_2), \ldots, S_2(t_N)\right), 0\right)\right] \tag{37}$$

where $S_1(t_i)$ and $S_2(t_i)$ are the asset prices at time t_i for the original and antithetic variate paths respectively and $\mathbb{E}^{\mathbb{Q}}$ is the expectation under the risk-neutral measure. The same equations can be used for the fixed strike minimum lookback options by replacing the maximum with a minimum.

3.2.4 Antithetic Variates Implementation

We will now implement the antithetic variates technique by creating a new `AntitheticMonteCarloSimulation` class. We will add the method `run_Antithetic_GBM()` which will behave similarly to `run_GBM()` but instead simulate the underlying asset price paths using the Euler-Maruyama scheme with antithetic variates, it will be called using the same `generate_Paths()` method to maintain the same function signature. We will also create new classes for each of the options to inherit from this `AntitheticMonteCarloSimulation` class and implement their adjusted payoff calculation with the same function signature of `call_Price()` and `put_Price()`. This means we can create instances of the old and new classes and run their methods in the same way.

To test the implementation of the antithetic variates technique we will use the same sample data as before but run only half the number of total Monte Carlo simulations. We'll then compare the option prices and standard errors of the original and antithetic variate classes.

In [20]:
```
# Half the number of simulations
NUMBER_OF_ANTITHETIC_SIMULATIONS = int(NUMBER_OF_SIMULATIONS / 2)
```

In [21]:
```
class AntitheticMonteCarloSimulation:
    ''' Class for an antithetic Monte Carlo simulation using the Euler-Maruyama
    scheme to simulate the underlying asset price paths.
    Args:
        S0 (float): The initial underlying asset price
        E (float): The strike price
        T (float): The time horizon or time to maturity in years
        r (float): Risk-free rate as the drift coefficient
        sigma (float): Volatility
        number_of_paths (int): The number of path realisations
        random_seed (int): The random seed for reproducibility [default: 0]

    Returns:
        None
    '''
    def __init__(self, S0: float, E: float, T: float, r: float,
                 sigma: float, number_of_paths: int, random_seed: int):
        self.S0 = S0
        self.E = E
        self.time_Steps = int(T * TRADING_DAYS_PER_YEAR)
        self.T = T
        self.r = r
        self.sigma = sigma
        self.number_Of_Paths = number_of_paths
        self.random_Seed = random_seed
```

```python
def run_Antithetic_GBM(self) -> Tuple[np.ndarray, np.ndarray, np.ndarray]:
    ''' Simulate the underlying asset price paths and their antithetic variate
    paths assuming they follow a risk-neutral Geoemtric Brownian Motion
    process using the Euler-Maruyama scheme.
    Args:
        None

    Returns:
        St_1 (np.ndarray): An array of the simulated asset price paths with
        shape (time_steps, number_of_paths)
        St_2 (np.ndarray): An array of the simulated antithetic variate asset
        price paths with shape (time_steps, number_of_paths)
        time_axis (np.ndarray): An array of the time axis used for plotting
        with shape (time_steps, )
    '''
    # Set the random seed
    np.random.seed(self.random_Seed)

    # Calculate dt and time axis used for plotting
    T = self.T

    time_axis, dt = np.linspace(0, T, self.time_Steps, retstep=True)

    # Set the input paramters, r is the risk-free rate
    r = self.r
    sigma = self.sigma

    # Simulate paths
    St_1 = np.zeros((self.time_Steps, self.number_Of_Paths))
    St_1[0] = self.S0

    St_2 = np.zeros((self.time_Steps, self.number_Of_Paths))
    St_2[0] = self.S0

    # Produce an array of standard normal random numbers
    phi = np.random.standard_normal(size=(self.time_Steps,
                                          self.number_Of_Paths))

    # Simulate the GBM process and its antithetic variate using the
    # Euler-Maruyama scheme
    for t in range(1, self.time_Steps):
        St_1[t] = St_1[t-1] * np.exp((r - 0.5 * sigma**2) * dt + sigma
                                     * np.sqrt(dt) * phi[t-1])
        St_2[t] = St_2[t-1] * np.exp((r - 0.5 * sigma**2) * dt - sigma
                                     * np.sqrt(dt) * phi[t-1])

    return St_1, St_2, time_axis

def generate_Paths(self):
    ''' Run the Monte Carlo simulation using antithetic variates, storing
    the paths and time axis arrays in the St and tt attributes.'''
    # Time the simulation
    start = time.time()

    # Run the simulation
```

```
        self.St_1, self.St_2, self.time_Axis = self.run_Antithetic_GBM()

        # Calculate the time taken for the simulation
        self.run_Time = time.time() - start
```

We'll have to re-define each of the option classes to inherit from the `MonteCarloSimulation` class with antithetic variates. We will then re-calculate the option prices for the Asian and lookback options using the antithetic variates technique.

In [22]:
```
class AntitheticAsianOption(AntitheticMonteCarloSimulation):
    def __init__(self, S0: float, E: float, T: float, r: float,
                 sigma: float, number_of_paths: int, random_seed):
        super().__init__(S0, E, T, r, sigma, number_of_paths, random_seed)

    def call_Price(self, standard_error: bool = False) -> float:
        ''' Calculate the price of a fixed strike arithmetic Asian call option
        with time to maturity T by discounting the average payoff
        Args:
            standard_error (bool): Whether to calculate the standard error of the
            call option price

        Returns:
            C (float): The price of the call option under the risk-neutral measure
        '''
        # Parameters for readability
        St_1 = self.St_1
        St_2 = self.St_2
        E = self.E
        T = self.T
        r = self.r

        # Calculate the arithmetic average (A) of the underlying asset prices
        A_1 = np.mean(St_1, axis=0)
        A_2 = np.mean(St_2, axis=0)

        # Calculate the avergae payoff of each path and its antithetic variate
        CT = (np.maximum(A_1 - E, 0) + np.maximum(A_2 - E, 0)) / 2

        # Calculate the price of the call option
        C = np.exp(-r * T) * np.mean(CT)

        if standard_error:
            # Calculate the standard error of the call option price
            SE = np.std(CT) / np.sqrt(self.number_Of_Paths)
            return C, SE

        else:
            return C

    def put_Price(self, standard_error: bool = False) -> float:
        ''' Calculate the price of a fixed strike arithmetic Asian put option with
        time to maturity T by discounting the average payoff
        Args:
            standard_error (bool): Whether to calculate the standard error of the
            call option price
```

```
        Returns:
            C (float): The price of the put option under the risk-neutral measure
        '''
        # Parameters for readability
        St_1 = self.St_1
        St_2 = self.St_2
        E = self.E
        T = self.T
        r = self.r

        # Calculate the arithmetic average (A) of the underlying asset prices
        A_1 = np.mean(St_1, axis=0)
        A_2 = np.mean(St_2, axis=0)

        # Calculate the avergae payoff of each path and its antithetic variate
        PT = (np.maximum(E - A_1, 0) + np.maximum(E - A_2, 0)) / 2

        # Calculate the price of the put option
        P = np.exp(-r * T) * np.mean(PT)

        if standard_error:
            # Calculate the standard error of the put option price
            SE = np.std(PT) / np.sqrt(self.number_Of_Paths)
            return P, SE

        else:
            return P
```

In [23]:

```
class AntitheticMaximumLookbackOption(AntitheticMonteCarloSimulation):
    def __init__(self, S0: float, E: float, T: float, r: float,
                 sigma: float, number_of_paths: int, random_seed):
        super().__init__(S0, E, T, r, sigma, number_of_paths, random_seed)

    def call_Price(self, standard_error: bool = False) -> float:
        ''' Calculate the price of a fixed strike lookback call option with
        time to maturity T by discounting the discretely sampled maximum payoff
        Args:
            standard_error (bool): Whether to calculate the standard error of the
            call option price

        Returns:
            C (float): The price of the call option under the risk-neutral measure
        '''
        # Parameters for readability
        St_1 = self.St_1
        St_2 = self.St_2
        E = self.E
        T = self.T
        r = self.r

        # Calculate the maximum (M) of the underlying asset prices
        M_1 = np.max(St_1, axis=0)
        M_2 = np.max(St_2, axis=0)

        # Calculate the avergae payoff of each path and its antithetic variate
```

```
        CT = (np.maximum(M_1 - E, 0) + np.maximum(M_2 - E, 0)) / 2

        # Calculate the price of the call option
        C = np.exp(-r * T) * np.mean(CT)

        if standard_error:
            # Calculate the standard error of the call option price
            SE = np.std(CT) / np.sqrt(self.number_Of_Paths)
            return C, SE

        else:
            return C

    def put_Price(self, standard_error: bool = False) -> float:
        ''' Calculate the price of a fixed strike lookback put option with
        time to maturity T by discounting the discretely sampled maximum payoff
        Args:
            standard_error (bool): Whether to calculate the standard error of the
            call option price

        Returns:
            P (float): The price of the put option under the risk-neutral measure
        '''
        # Parameters for readability
        St_1 = self.St_1
        St_2 = self.St_2
        E = self.E
        T = self.T
        r = self.r

        # Calculate the maximum (M) of the underlying asset prices
        M_1 = np.max(St_1, axis=0)
        M_2 = np.max(St_2, axis=0)

        # Calculate the avergae payoff of each path and its antithetic variate
        PT = (np.maximum(E - M_1, 0) + np.maximum(E - M_2, 0)) / 2

        # Calculate the price of the put option
        P = np.exp(-r * T) * np.mean(PT)

        if standard_error:
            # Calculate the standard error of the put option price
            SE = np.std(PT) / np.sqrt(self.number_Of_Paths)
            return P, SE

        else:
            return P
```

In [24]:

```
class AntitheticMinimumLookbackOption(AntitheticMonteCarloSimulation):
    def __init__(self, S0: float, E: float, T: float, r: float,
                 sigma: float, number_of_paths: int, random_seed):
        super().__init__(S0, E, T, r, sigma, number_of_paths, random_seed)

    def call_Price(self, standard_error: bool = False) -> float:
        ''' Calculate the price of a fixed strike minimum lookback call option with
        time to maturity T by discounting the discretely sampled minimum payoff
```

```
        Args:
            standard_error (bool): Whether to calculate the standard error of the
            call option price

        Returns:
            C (float): The price of the call option under the risk-neutral measure
        '''
        # Parameters for readability
        St_1 = self.St_1
        St_2 = self.St_2
        E = self.E
        T = self.T
        r = self.r

        # Calculate the minimum (M) of the underlying asset prices
        M_1 = np.min(St_1, axis=0)
        M_2 = np.min(St_2, axis=0)

        # Calculate the avergae payoff of each path and its antithetic variate
        CT = (np.maximum(M_1 - E, 0) + np.maximum(M_2 - E, 0)) / 2

        # Calculate the price of the call option
        C = np.exp(-r * T) * np.mean(CT)

        if standard_error:
            # Calculate the standard error of the call option price
            SE = np.std(CT) / np.sqrt(self.number_Of_Paths)
            return C, SE

        else:
            return C

    def put_Price(self, standard_error: bool = False) -> float:
        ''' Calculate the price of a fixed strike minimum lookback put option with
        time to maturity T by discounting the discretely sampled minimum payoff
        Args:
            standard_error (bool): Whether to calculate the standard error of the
            call option price

        Returns:
            P (float): The price of the put option under the risk-neutral measure
        '''
        # Parameters for readability
        St_1 = self.St_1
        St_2 = self.St_2
        E = self.E
        T = self.T
        r = self.r

        # Calculate the minimum (M) of the underlying asset prices
        M_1 = np.min(St_1, axis=0)
        M_2 = np.min(St_2, axis=0)

        # Calculate the avergae payoff of each path and its antithetic variate
        PT = (np.maximum(E - M_1, 0) + np.maximum(E - M_2, 0)) / 2
```

```
        # Calculate the price of the put option
        P = np.exp(-r * T) * np.mean(PT)

        if standard_error:
            # Calculate the standard error of the put option price
            SE = np.std(PT) / np.sqrt(self.number_Of_Paths)
            return P, SE

        else:
            return P
```

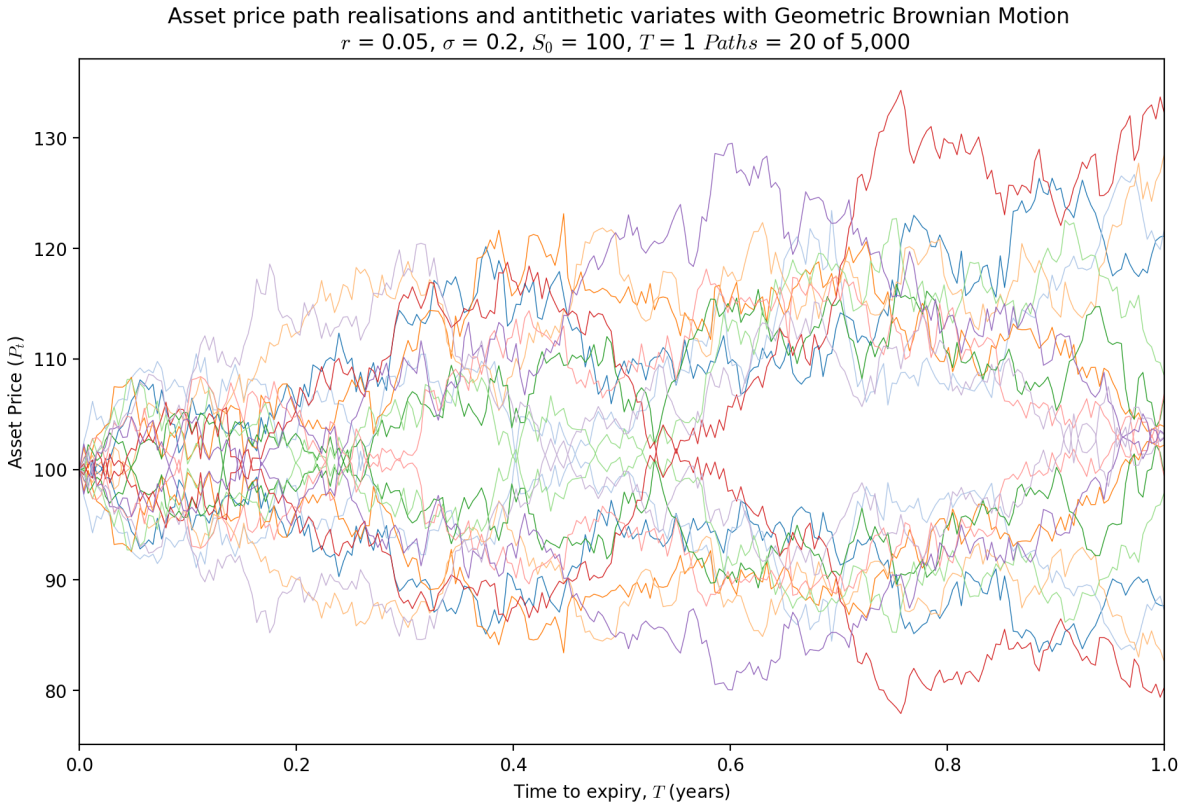
3.2.5 Visualising the Antithetic Variate Paths

The below plot shows the simulated asset price paths and their antithetic variate paths with the intial input data using 5,000 path realisations of the underlying asset price over the time horizon. We've plotted 20 of the 5,000 simulated asset price paths for clarity purposes.

Each path and it's antithetic variate are plotted in the same colour. We can see that the antithetic variate paths are the mirror image of the original paths. This is because the antithetic variate paths are generated using the negative of the original set of Normal random numbers $-\phi_i$.

In [25]:
```
# Initialise parameters
S0 = 100
E = 100
T = 1
sigma = 0.2
r = 0.05

# Get each of the option objects
asset = AntitheticMonteCarloSimulation(S0, E, T, r,
                                       sigma, NUMBER_OF_ANTITHETIC_SIMULATIONS,
                                       random_seed)
asset.generate_Paths()

# Plot 10 random paths and their antithetic variates
plt.figure(figsize=(11, 7))

# Generate a list of unique colours
unique_colours = plt.cm.tab20.colors
colour_iterator = iter(unique_colours)

display_paths = 10
for i in np.random.choice(np.array(range(NUMBER_OF_ANTITHETIC_SIMULATIONS)),
                          display_paths):
    colour = next(colour_iterator)
    plt.plot(asset.time_Axis, asset.St_1[:, i], color=colour, lw=0.5)
    plt.plot(asset.time_Axis, asset.St_2[:, i], color=colour, lw=0.5)

plt.xlim(0, T)
plt.xlabel('Time to expiry, $T$ (years)')
plt.ylabel('Asset Price ($P_t$)')
title = f'Asset price path realisations and antithetic variates with Geometric ' \
```

```
        f'Brownian Motion \n $r$ = {r}, $\sigma$ = {sigma}, $S_0$ = {S0}, $T$ = {T}
        f'$Paths$ = {display_paths*2:,d} of {NUMBER_OF_ANTITHETIC_SIMULATIONS:,d}'
plt.title(title)
plt.show()
```



3.2.6 Initial Sample Data Results using Antithetic Variates

Let's re-run the initial sample data using the antithetic variates technique and compare the run-time and standard error of the simulations. We'll update the `OptionFactory` class to include the antithetic variates option type and re-run the initial sample data using both the original and the antithetic variates technique.

$$\begin{aligned}\text{Today's stock price } S_0 &= 100\\ \text{Strike } E &= 100\\ \text{Time to expiry } (T-t) &= 1 \text{ year}\\ \text{volatility } \sigma &= 20\%\\ \text{constant risk-free interest rate } r &= 5\%\end{aligned}$$

In [26]:
```
class OptionFactory:
    ''' Factory class for creating option objects'''

    @staticmethod
    def create_Option(option_type: str, S0: float, E: float, T: float, r: float,
                      sigma: float, random_seed: int):
        ''' Return an option object based on the option type
        Args:
            option_type (str): The type of option to create
            S0 (float): The initial underlying asset price
```

```
            E (float): The strike price
            T (float): The time horizon or time to maturity in years
            r (float): Risk-free rate as the drift coefficient
            sigma (float): Volatility
            number_of_paths (int): The number of path realisations
            random_seed (int): The random seed for reproducibility [default: 0]

        Returns:
            option (object): An option object
        '''
        if option_type == 'Asian':
            return AsianOption(S0, E, T, r, sigma,
                               NUMBER_OF_SIMULATIONS, random_seed)
        elif option_type == 'Asian antithetic':
            return AntitheticAsianOption(S0, E, T, r, sigma,
                                         NUMBER_OF_ANTITHETIC_SIMULATIONS,
                                         random_seed)
        elif option_type == 'Maximum lookback':
            return MaximumLookbackOption(S0, E, T, r, sigma,
                                         NUMBER_OF_SIMULATIONS,
                                         random_seed)
        elif option_type == 'Maximum lookback antithetic':
            return AntitheticMaximumLookbackOption(S0, E, T, r, sigma,
                                                   NUMBER_OF_ANTITHETIC_SIMULATIONS
                                                   random_seed)
        elif option_type == 'Minimum lookback':
            return MinimumLookbackOption(S0, E, T, r, sigma,
                                         NUMBER_OF_SIMULATIONS, random_seed)
        elif option_type == 'Minimum lookback antithetic':
            return AntitheticMinimumLookbackOption(S0, E, T, r, sigma,
                                                   NUMBER_OF_ANTITHETIC_SIMULATIONS
                                                   random_seed)
        else:
            raise ValueError('Option type not recognised')
```

In [27]:

```
# Initialise parameters
S0 = 100
E = 100
T = 1
sigma = 0.2
r = 0.05

# Random seed for reproducibility
random_seed = 0

options = ['Asian', 'Maximum lookback', 'Minimum lookback', 'Asian antithetic',
           'Maximum lookback antithetic', 'Minimum lookback antithetic']
results = []

for option_type in options:
    # Create an option object and generate paths
    option = OptionFactory.create_Option(option_type, S0, E, T, r,
                                         sigma, random_seed)
    option.generate_Paths()

    call = option.call_Price(standard_error=True)
```

```
    put = option.put_Price(standard_error=True)
    call_price, call_SE, put_price, put_SE = call[0], call[1], put[0], put[1]

    results.append([f'{option_type} call', call_price, call_SE,
                    option.number_Of_Paths, option.run_Time])
    results.append([f'{option_type} put', put_price, put_SE,
                    option.number_Of_Paths, option.run_Time])

headers = ['Option', 'Price', 'SE +/-', 'Paths', 'Run Time']
print(tabulate.tabulate(results,
                        headers=headers,
                        floatfmt=('.1f', '.2f', '.3f', '.1f', '.3f'),
                        tablefmt='psql'))
```

```
+---------------------------------+---------+----------+---------+------------+
| Option                          |   Price |   SE +/- |   Paths |   Run Time |
|---------------------------------+---------+----------+---------+------------|
| Asian call                      |    5.76 |    0.083 |   10000 |      0.053 |
| Asian put                       |    3.21 |    0.054 |   10000 |      0.053 |
| Maximum lookback call           |   18.40 |    0.161 |   10000 |      0.052 |
| Maximum lookback put            |    0.00 |    0.000 |   10000 |      0.052 |
| Minimum lookback call           |    0.00 |    0.000 |   10000 |      0.049 |
| Minimum lookback put            |   11.51 |    0.093 |   10000 |      0.049 |
| Asian antithetic call           |    5.73 |    0.057 |    5000 |      0.034 |
| Asian antithetic put            |    3.33 |    0.042 |    5000 |      0.034 |
| Maximum lookback antithetic call |   18.16 |    0.093 |    5000 |      0.032 |
| Maximum lookback antithetic put  |    0.00 |    0.000 |    5000 |      0.032 |
| Minimum lookback antithetic call |    0.00 |    0.000 |    5000 |      0.032 |
| Minimum lookback antithetic put  |   11.62 |    0.051 |    5000 |      0.032 |
+---------------------------------+---------+----------+---------+------------+
```

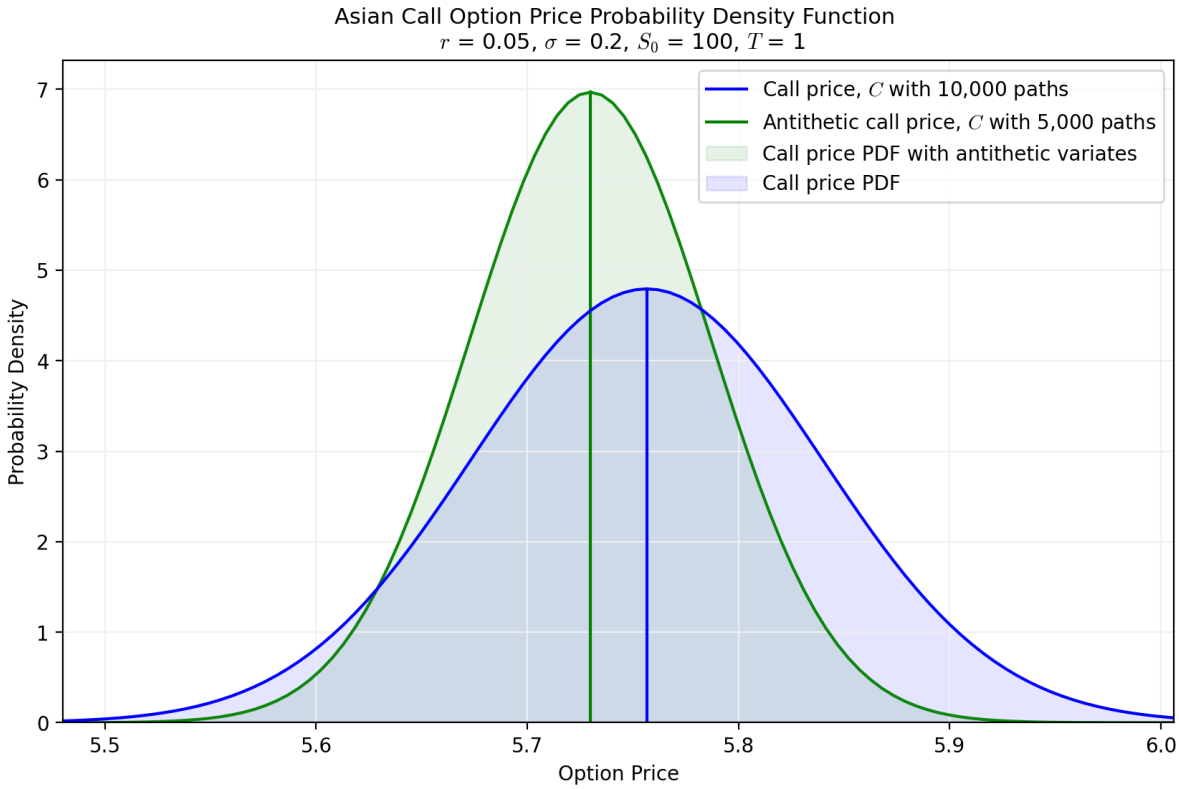
As we can see, the standard error for each option price using the antithetic variate technique is smaller than that of the original technique despite running only half the number of simulations. The run time is considerably faster due to running half as many simulations, yet we're still able to maintain an acceptable standard error.

3.2.7 Visualising Convergence of the Antithetic Variate Technique

The below plot shows the spread of the option price estimates around the expected value of the option price for the Asian call option using the original and antithetic variate technique. The option price is calculated using 10,000 path realisations of the underlying asset price over the time horizon and 5,000 path realisations using the antithetic variate technique.

In [28]:
```
# Create Asian and antithetic Asian option objects
asian_option = OptionFactory.create_Option('Asian', S0, E, T, r,
                                           sigma, random_seed)
antithetic_asian_option = OptionFactory.create_Option('Asian antithetic',
                                                      S0, E, T, r,
                                                      sigma, random_seed)

# Generate paths for each option
asian_option.generate_Paths()
antithetic_asian_option.generate_Paths()
```

```
# Calculate the call price and standard error for each option
asian_C, asian_call_SE = asian_option.call_Price(standard_error=True)
antithetic_asian_C, \
antithetic_asian_call_SE = antithetic_asian_option.call_Price(standard_error=True)

# Generate the option price x axis
x = np.linspace(min(asian_C, antithetic_asian_C) - 3 *
                max(asian_call_SE, antithetic_asian_call_SE),
                max(asian_C, antithetic_asian_C) + 3 *
                max(asian_call_SE, antithetic_asian_call_SE), 100)

# Calculate the probability density functions (PDFs) for the normal distributions
pdf_asian = ss.norm.pdf(x, asian_C, asian_call_SE)
pdf_antithetic_asian = ss.norm.pdf(x, antithetic_asian_C, antithetic_asian_call_SE)

# Create the plot
plt.figure(figsize=(10, 6))

# # Plot the Asian call price
plt.plot([asian_C, asian_C], [0, max(pdf_asian)], 'b',
         label=f'Call price, $C$ with {NUMBER_OF_SIMULATIONS:,d} paths')

# Plot the Asian antithetic call price
plt.plot([antithetic_asian_C, antithetic_asian_C], [0, max(pdf_antithetic_asian)],
         'g', label=f'Antithetic call price, $C$ with ' \
         f'{NUMBER_OF_ANTITHETIC_SIMULATIONS:,d} paths')

# Plot the PDFs
plt.plot(x, pdf_antithetic_asian, '#0C860C')
plt.fill_between(x, pdf_antithetic_asian, color='#0C860C', alpha=0.1,
                 label='Call price PDF with antithetic variates')
plt.plot(x, pdf_asian, 'b')
plt.fill_between(x, pdf_asian, color='b', alpha=0.1, label='Call price PDF')

plt.xlim(min(x), max(x))
plt.ylim(0)
plt.xlabel('Option Price')
plt.ylabel('Probability Density')
plt.legend()
title = f'Asian Call Option Price Probability Density Function \n ' \
        f'$r$ = {r}, $\sigma$ = {sigma}, $S_0$ = {S0}, $T$ = {T}'
plt.title(title, fontsize=11)
plt.grid()

plt.show()
```



4. Conclusion

In this report we've discussed the pricing of exotic options, specifically, fixed strike Asian and lookback options using Euler-Maruyama scheme with a Monte Carlo pricing method. We've seen that the Monte Carlo method is a very powerful tool for pricing options, especially in higher dimensions where there is strong path dependency, however it is computationally expensive and due to reliance on generating a finite number of realisations using random numbers, the option price is subject to error and will vary with each run of the simulation.

By varying the input data we've seen that the option price is sensitive to the initial asset price, S_0, the time to expiry, T, and the volatility, σ, however the degree of sensitivity depends on the type of option and whether it is in the money or out of the money. Changing S_0 resulted in higher call option prices and lower put option prices for all options. Time to expiry, T, had little effect on the Asian option prices as the average asset price is calculated over the life of the option, conversely for lookback options the option prices were higher when the option was in the money with a small T. In terms of changing volatility, σ, this had a large effect on all option prices as the underlying asset price was more likely to move further away from the strike price, E, with a higher volatility.

By acknowleding the speed and standard error drawbacks of the Monte Carlo method we then sought to explore how to speed up the convergence of the Monte Carlo simulations by using variance reduction techniques (such as antithetic variates) as oppose to increasing the number of simulations. As shown in the plot in the previous section we found that the

antithetic variates technique reduces the standard error of the option price and therefore the number of simulations required to achieve a given accuracy.

5. References

- Wilmott, P. (2007) ‘11, 27, 29’, in Paul Wilmott introduces Quantitative Finance. 2nd edn. Chichester, West Sussex, England: J. Wiley & Sons, pp. 252-253, 265, 544-545, 581-598.
- Wilmott, P. (2006) ‘22, 24, 25, 26’, in Paul Wilmott On Quantitative Finance. 2nd edn. Chichester, England: John Wiley & Sons, pp. 367-384, 417–445.
- Wilmott, P. (2006b) ‘80’, in Paul Wilmott On Quantitative Finance. Chichester, England: John Wiley & Sons, pp. 1263–1278.
- Clewlow, L. and Strickland, C. (2007) ‘4’, in Implementing Derivatives Models. Chichester: Wiley, pp. 82–96.
- Singaravelu, K. (2023) Monte Carlo Option Pricing Python Lab
- Zhang, Hongbin. Pricing Asian Options using Monte Carlo Methods. 2009.