1. Finance Problem Summary

Use the expected value of the discounted payoff under the risk-neutral density $\mathbb{Q}$:

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

for the appropriate form of payoff, to consider ***Asian*** and ***lookback*** options.

Use the ***Euler-Maruyama*** only scheme for initially simulating the underlying stock price. As an initial example you may use the following set of sample data:

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

Then vary the data to see the affect on the option price.

2. Solution - Numerical Procedure

2.1 Monte Carlo Approach

In this case, I will use Monte Carlo approach to simulate the risk neutral random walk. Generally speaking, this method usually consists of three steps:

Step 1: Simulate sample stock paths under the risk-neutral measure.
Step 2: Calculate the discounted payoff on each simulated path.
Step 3: Average the payoff and discount back to today to determine the option price.

2.2 Simulating Stock Prices Path using Euler-Maruyama Scheme

Then, we'll simulate the stock price at maturity S_T. Based on the Monte Carlo Simulation Python Labs, we know that a geometric Brownian motion with a stochastic differential equation (SDE) is given as below following the Black-Scholes-Merton where the underlying follows under the risk neutrality:

$$dS_t = rS_t dt + \sigma S_t dZ_t$$

where S_t is the price of the underlying at time t, σ is constant volatility, r is the constant risk-free interest rate and Z is the Brownian motion.

Applying Euler-Maruyama discretization of SDE, we get:

$$S_{t+\Delta t} = S_t * (1 + r\Delta t + \sigma\sqrt{\Delta t}z)$$

It is often more convenient to express in time stepping form

$$S_{t+\Delta t} = S_t \exp^{((r-\frac{1}{2}\sigma^2)\Delta t + \sigma\sqrt{\Delta t}z)}$$

The variable z is a standard normally distributed random variable, $0 < \Delta t < T$, time interval. It also holds $0 < t \leq T$ with T the final time horizon.

2.2.1 Import libraries

In [ ]:
```
# Import libraries
import pandas as pd
import numpy as np
from numpy import *
from tabulate import tabulate
import matplotlib.pyplot as plt
import cufflinks as cf
cf.set_config_file(offline=True)
# plt.style.use('seaborn-whitegrid')
plt.style.use("seaborn-v0_8-whitegrid")
from src.exotic_options import ExoticOptions
```

2.2.2 Simulating paths

In [ ]:
```
# define simulation function
def simulate_path(s0, risk_free_rate, vol, horizon, timesteps, n_sims):

    # set the random seed for reproducibility
    random.seed(2023)

    # read parameters
    S0 = s0                 # initial spot price
    r = risk_free_rate
    T = horizon             # time horizon
    t = timesteps           # number of time steps
    n = n_sims              # number of simulation
    vol = vol

    # define dt
    dt = T/t                # length of time interval

    # simulate 'n' asset price path with 't' timesteps
    S = zeros((t,n))
    S[0] = S0

    for i in range(0, t-1):
        w = random.standard_normal(n)
        S[i+1] = S[i] * np.exp((r - 0.5 * vol**2)*dt + vol*np.sqrt(dt)*w)

    return S
```

In [ ]:
```
# Define parameters and variables
s0 = 100
E = 100
vol = 0.2
T = 1
r = 0.05
```

```
In [ ]: # Define simulation parameters
n = 100000
t = 252
```

```
In [ ]: # Generating price paths
price_path = pd.DataFrame(simulate_path(s0, r, vol, T, t, n))
price_path
```

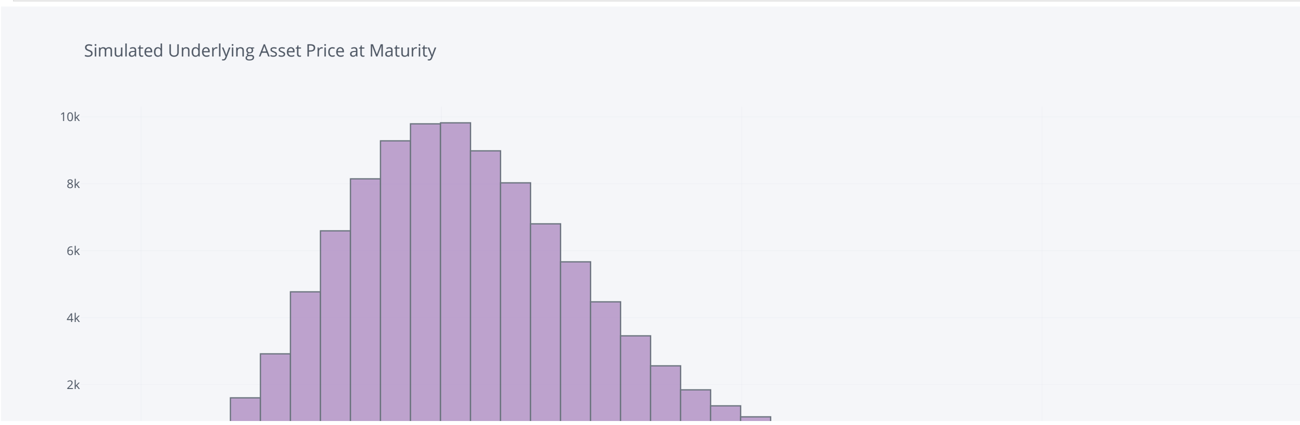
Out[ ]:

	0	1	2	3	4	5	6	7	8	9	...	99990	99991	99992	99993	99994	99995
0	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	...	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000
1	100.912669	99.603879	98.757451	100.310032	99.883264	98.584130	103.413097	101.843687	100.136603	96.155015	...	100.519944	98.858864	98.836039	100.031524	98.911140	99.573136
2	101.090391	99.238513	98.054709	100.310851	98.596189	100.734990	104.220766	100.236442	101.461604	95.092404	...	101.545778	97.150429	99.121461	100.236021	97.616665	99.459495
3	100.554081	100.367374	98.138477	100.281708	100.970674	101.013811	105.080348	96.880139	100.572892	93.877178	...	100.673720	95.680984	98.848203	100.315768	98.254332	101.351636
4	100.078638	100.458778	99.209790	100.309825	100.913121	100.440571	105.005227	96.928807	99.189835	92.919194	...	103.608113	96.964256	100.480481	99.695405	98.222164	100.701224
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
247	82.507624	76.504565	96.076824	104.906150	106.347956	144.724588	107.192059	87.646457	146.684418	98.843559	...	104.603689	97.572180	102.666292	129.297039	85.191158	163.500954
248	83.683408	75.719806	96.543515	103.535060	106.441828	146.847463	108.368876	86.602922	146.034397	98.900845	...	104.303403	96.453768	104.285182	129.018480	83.364000	164.767927
249	82.080723	76.865817	96.230818	103.485482	107.293378	150.081199	107.627991	85.647737	148.047691	99.307918	...	105.716184	96.107829	102.876678	128.278199	84.203958	164.383816
250	80.712146	76.670468	95.090290	105.557333	107.918073	149.780245	107.329224	87.098121	148.643154	98.744973	...	107.501375	94.822884	102.955574	125.626280	83.576562	163.546824
251	80.846035	76.595033	94.621059	103.541299	109.054810	151.335755	107.492276	86.195341	147.218928	100.523944	...	107.752150	96.796071	102.852081	122.937012	83.600300	164.004086

252 rows × 100000 columns

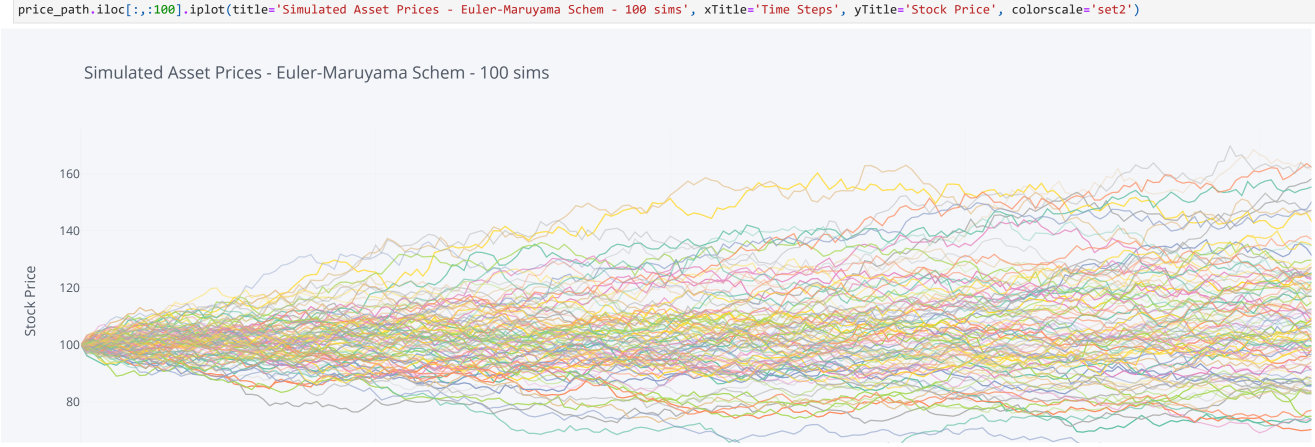
2.2.3 Histogram of Simulated Paths

```
In [ ]: # Plot the histogram of the simulated price path at maturity
price_path.iloc[-1].iplot(kind='histogram', title= 'Simulated Underlying Asset Price at Maturity', bins=100, colorscale='prgn')
```

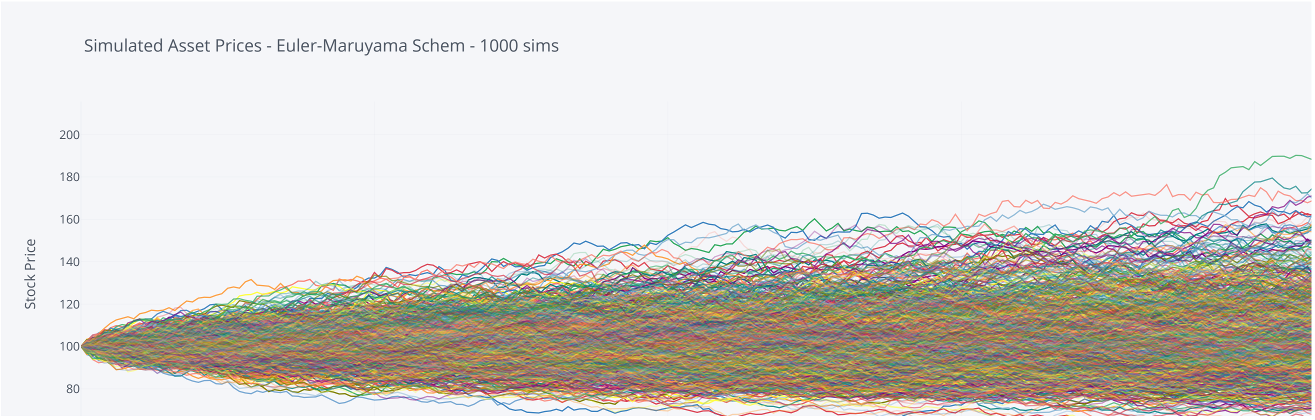


2.2.4 Visualization of Simulated Paths

```
In [ ]: # Plot simulated price paths, 1-100th
price_path.iloc[:,:100].iplot(title='Simulated Asset Prices - Euler-Maruyama Schem - 100 sims', xTitle='Time Steps', yTitle='Stock Price', colorscale='set2')
```




```
In [ ]: # Plot simulated price paths, 1-1000th
price_path.iloc[:,:1000].iplot(title='Simulated Asset Prices - Euler-Maruyama Schem - 1000 sims', xTitle='Time Steps', yTitle='Stock Price')
```



2.3 Exotic Options Pricing - Asian Options

2.3.1 Definition, Payoff types, Average methods

An Asian option is a special type of option contract, where the payoff is based on the average of prices of underlying asset over some period prior to maturity. There are two types of Asian options in financial markets: fixed strike, where averaging price is used in place of underlying price; and fixed price, where averaging price is used in place of strike. Knowing that the Asian options payoff depends on the average of the underlying stock price, we can average discretely or continuously, in it's arithmetic or geometric form - so two methods to calculate the average of prices.

Assuming A the average of the stock price, the payoffs are given by:

- **Average strike call**

$$max(S - A, 0)$$

- **Average strike put**

$$max(A - S, 0)$$

- **Average rate call**

$$max(A - E, 0)$$

- **Average rate put**

$$max(E - A, 0)$$

where S is the stock price and E the strike.

2.3.2 Arithmetic Averages - Continuous & Discrete

In this session, I will simplify the calculation by setting the new number of simulations ***n_new*** to 100.

In [ ]:
```
n_new = 100
S = ExoticOptions.monte_carlo_simulation(s0,r,vol,T,t,n_new)
S_Path = pd.DataFrame(S)
S_Path
```

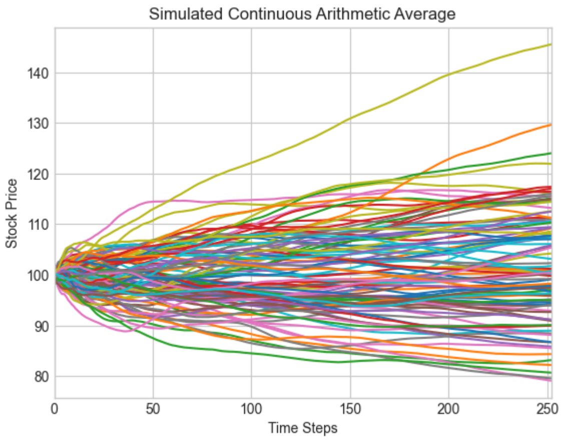
Out[ ]:

	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93	94	95
0	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	...	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000
1	98.423047	100.234090	99.639359	100.424696	98.792857	100.299203	98.984836	100.147101	101.248367	100.115249	...	101.589816	101.622007	97.890961	100.871961	100.720542	96.829023
2	98.520923	100.877066	99.020559	99.465684	100.553289	99.381340	98.833509	101.101772	100.774543	99.928186	...	101.494534	100.143772	98.771642	104.354779	100.232488	97.629788
3	99.207605	102.225339	97.607414	99.106927	100.921641	100.002798	98.231294	102.176024	99.042767	100.041161	...	100.975222	103.054794	99.527756	103.045720	101.155981	96.282441
4	96.453947	104.202245	97.463334	99.575957	100.866511	98.266960	96.248945	101.244402	99.444793	99.068526	...	100.425498	101.942977	98.702195	103.754679	101.440204	96.085120
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
248	88.040465	93.169453	93.324024	112.615639	102.382190	95.630586	77.082072	103.303765	114.524838	117.850339	...	100.724239	103.426199	92.463059	123.975077	126.741567	91.435903
249	88.544638	94.074459	94.632714	111.953237	103.136778	95.664095	76.680607	103.199251	120.238928	116.767630	...	100.241958	103.529163	92.566935	123.278812	125.298489	92.104176
250	87.525251	91.903819	96.134745	111.113320	100.305629	97.003527	77.776147	100.672493	119.795805	116.687923	...	100.430903	104.023328	93.918527	122.721173	125.883772	92.593612
251	89.242173	92.028793	94.568729	111.191963	100.520437	96.344187	77.645876	102.954171	118.930852	117.761007	...	101.778271	103.948981	91.956143	121.752272	125.460608	92.501453
252	90.466073	90.788034	94.354632	111.804277	102.056876	97.493512	77.562525	102.624764	120.298093	118.414978	...	100.850375	102.963795	91.425604	122.156944	125.883473	94.824889

253 rows × 100 columns

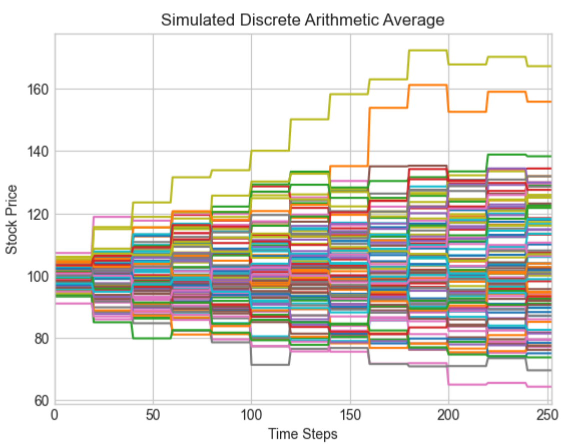
In [ ]:
```
# Continuous Arithmetic Average
AC = {}

AC = ExoticOptions.continuous_arithmetic_avrg(AC, S_Path)

Continuous_Arithmetic_Average = pd.DataFrame(AC)

for index in range(0, n_new, 1):
    plt.plot(Continuous_Arithmetic_Average[index])
    plt.grid(True)
    plt.xlabel('Time Steps')
    plt.xlim(0,t)
```

```
    plt.ylabel('Stock Price')
    plt.title('Simulated Continuous Arithmetic Average');
```



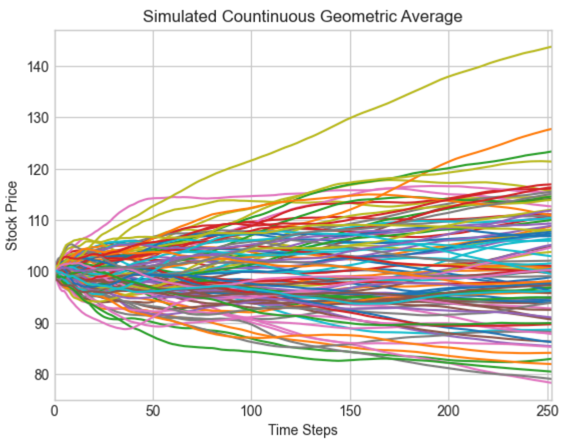
In [ ]:
```
# Define the average period used for discrete calculation
period = 20
```

In [ ]:
```
# Discrete Arithmetic Average
AD = {}

AD = ExoticOptions.discrete_arithmetic_avrg(AD, S_Path, period)

Discrete_Arithmetic_Average = pd.DataFrame(AD)

for index in range(0, n_new, 1):
    plt.plot(Discrete_Arithmetic_Average[index])
    plt.grid(True)
    plt.xlabel('Time Steps')
    plt.xlim(0,t)
    plt.ylabel('Stock Price')
    plt.title('Simulated Discrete Arithmetic Average');
```

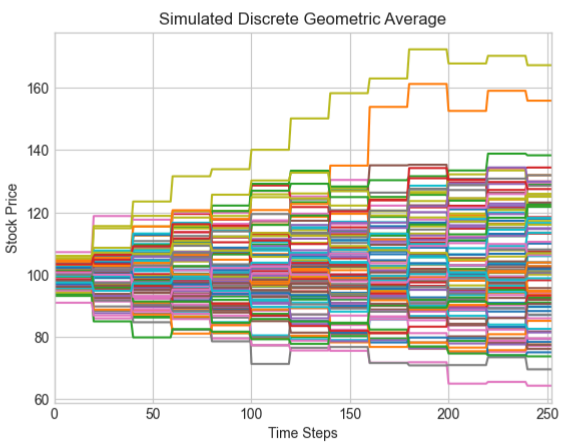


2.3.3 Geometric Averages - Continuous & Discrete

In [ ]:
```
# Countinuous Geometric Average
GC = {}

GC = ExoticOptions.continuous_geometric_avrg(GC, S_Path)

Countinuous_Geometric_Average = pd.DataFrame(GC)

for index in range(0, n_new, 1):
    plt.plot(Countinuous_Geometric_Average[index])
    plt.grid(True)
    plt.xlabel('Time Steps')
    plt.xlim(0,t)
    plt.ylabel('Stock Price')
    plt.title('Simulated Countinuous Geometric Average');
```



In [ ]:
```
# Discrete Geometric Average
GD = {}

GD = ExoticOptions.discrete_geometric_avrg(GD, S_Path, period)

Discrete_Geometric_Average = pd.DataFrame(GD)

for index in range(0, n_new, 1):
    plt.plot(Discrete_Geometric_Average[index])
    plt.grid(True)
    plt.xlabel('Time Steps')
    plt.xlim(0,t)
    plt.ylabel('Stock Price')
    plt.title('Simulated Discrete Geometric Average');
```



2.3.4 Asian Calls - Continuous vs Discrete; Strike vs Rate

In [ ]:
```
# Continuous Arithmetic Average Strike Call
caasc = ExoticOptions.payoff_mean(r, T, S_Path, Continuous_Arithmetic_Average)
# Continuous Geometric Average Strike Call
cgasc = ExoticOptions.payoff_mean(r, T, S_Path, Countinuous_Geometric_Average)
# Continuous Arithmetic Average Rate Call
caarc = ExoticOptions.payoff_mean(r, T, Continuous_Arithmetic_Average, E)
# Continuous Geometric Average Rate Call
cgarc = ExoticOptions.payoff_mean(r, T, Countinuous_Geometric_Average, E)


# Discrete Arithmetic Average Strike Call
daasc = ExoticOptions.payoff_mean(r, T, S_Path, Discrete_Arithmetic_Average)
# Discrete Geometric Average Strike Call
dgasc = ExoticOptions.payoff_mean(r, T, S_Path, Discrete_Geometric_Average)
# Discrete Arithmetic Average Rate Call
daarc = ExoticOptions.payoff_mean(r, T, Discrete_Arithmetic_Average, E)
# Discrete Geometric Average Rate Call
dgarc = ExoticOptions.payoff_mean(r, T, Discrete_Geometric_Average, E)
```

In [ ]:
```
print("Strike Call")

print(tabulate([["Continuous Arithmetic Average", caasc],
                ["Continuous Geometric Average", cgasc],
                ["Discrete Arithmetic Average", daasc],
                ["Discrete Geometric Average", dgasc]],
                floatfmt=".4f", tablefmt="fancy_grid"))

print("Rate Call")

print(tabulate([["Continuous Arithmetic Average", caarc],
                ["Continuous Geometric Average", cgarc],
                ["Discrete Arithmetic Average", daarc],
                ["Discrete Geometric Average", dgarc]],
                floatfmt=".4f", tablefmt="fancy_grid"))
```

Strike Call

Continuous Arithmetic Average	3.4717
Continuous Geometric Average	3.5646
Discrete Arithmetic Average	0.8561
Discrete Geometric Average	0.8682

Rate Call

Continuous Arithmetic Average	3.1783
Continuous Geometric Average	3.0863
Discrete Arithmetic Average	5.7719
Discrete Geometric Average	5.7583

2.3.5 Asian Puts - Continuous vs Discrete; Strike vs Rate

In [ ]:
```
# Continuous Arithmetic average Strike Put
caasp = ExoticOptions.payoff_mean(r, T, Continuous_Arithmetic_Average, S_Path)
# Continuous Geometric average Strike Put
cgasp = ExoticOptions.payoff_mean(r, T, Countinuous_Geometric_Average, S_Path)
# Continuous Arithmetic average Rate Put
caarp = ExoticOptions.payoff_mean(r, T, E, Continuous_Arithmetic_Average)
# Continuous Geometric average Rate Put
cgarp = ExoticOptions.payoff_mean(r, T, E, Countinuous_Geometric_Average)


# Discrete Arithmetic average Strike Put
daasp = ExoticOptions.payoff_mean(r, T, Discrete_Arithmetic_Average, S_Path)
# Discrete Geometric average Strike Put
dgasp = ExoticOptions.payoff_mean(r, T, Discrete_Geometric_Average, S_Path)
# Discrete Arithmetic average Rate Put
daarp = ExoticOptions.payoff_mean(r, T, E, Discrete_Arithmetic_Average)
# Discrete Geometric average Rate Put
dgarp = ExoticOptions.payoff_mean(r, T, E, Discrete_Geometric_Average)
```

In [ ]:
```
print("Strike Put")

print(tabulate([["Continuous Arithmetic Average", caasp],
                ["Continuous Geometric Average", cgasp],
                ["Discrete Arithmetic Average", daasp],
                ["Discrete Geometric Average", dgasp]],
                floatfmt=".4f", tablefmt="fancy_grid"))

print("Rate Put")

print(tabulate([["Continuous Arithmetic Average", caarp],
                ["Continuous Geometric Average", cgarp],
                ["Discrete Arithmetic Average", daarp],
                ["Discrete Geometric Average", dgarp]],
                floatfmt=".4f", tablefmt="fancy_grid"))
```

Strike Put

Continuous Arithmetic Average	2.1371
Continuous Geometric Average	2.0819
Discrete Arithmetic Average	0.8561
Discrete Geometric Average	0.8435

Rate Put

Continuous Arithmetic Average	2.7013
Continuous Geometric Average	2.7574
Discrete Arithmetic Average	3.9604
Discrete Geometric Average	3.9715

2.4 Exotic Options Pricing - Lookback Options

2.4.1 Definition, Payoff types, Average methods

Lookbacks are defined by the property that their expiry payoffs depend on the realized maximum or minimum of the asset price over a specified time window, called the lookback window. Lookback options, in the terminology of finance, are a type of exotic option with path dependency, among many other kind of options. The payoff depends on the optimal (maximum or minimum) underlying asset's price occurring over the life of the option.

Let's take maximum as an example here. Similarly, the maximum can be sampled continuously or discretely just like Asian options. Generally, there exist two kinds of lookback options: with **floating strike** and with **fixed strike**. These have payoffs that are the same as vanilla options except that in the strike option the vanilla exercise price is replaced by the maximum. In the rate option it is the asset value in the vanilla option that is replaced by the maximum.

Assuming M the maximum of the stock price, the payoffs are given by:

- **Lookback rate call**

$$max(M - E, 0)$$

- **Lookback rate put**

$$max(E - M, 0)$$

- **Lookback strike call**

$$max(S - M, 0)$$

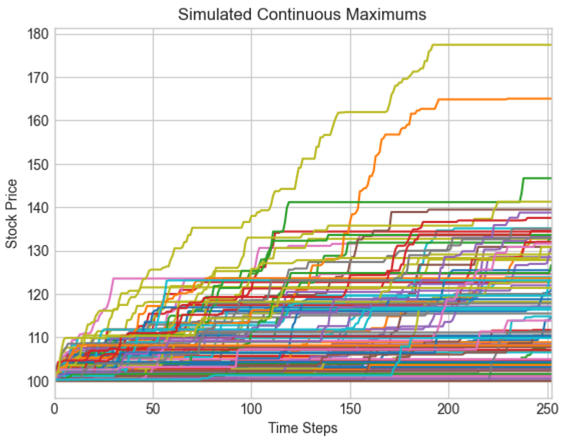
- **Lookback strike put**

$$max(M - S, 0)$$

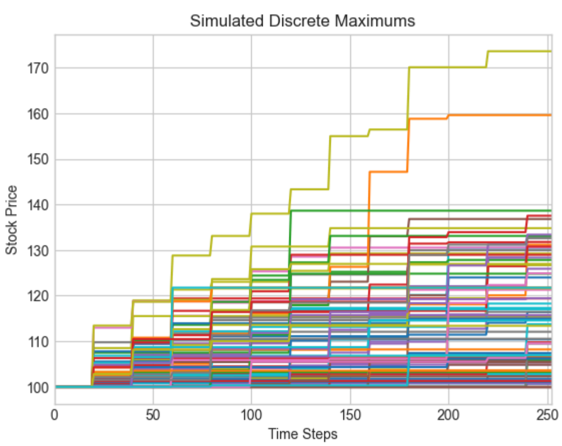
where S is the stock price and E the strike.

2.4.2 Continuous & Discrete Stock Maximum

In [ ]:
```
# Continuous Stock maximums
MC = {}

MC = ExoticOptions.continuous_maximum(MC, S_Path)

Continuous_max_stock = pd.DataFrame(MC)

for index in range(0, n_new, 1):
    plt.plot(Continuous_max_stock[index])
    plt.grid(True)
    plt.xlabel('Time Steps')
    plt.xlim(0,t)
    plt.ylabel('Stock Price')
    plt.title('Simulated Continuous Maximums');
```

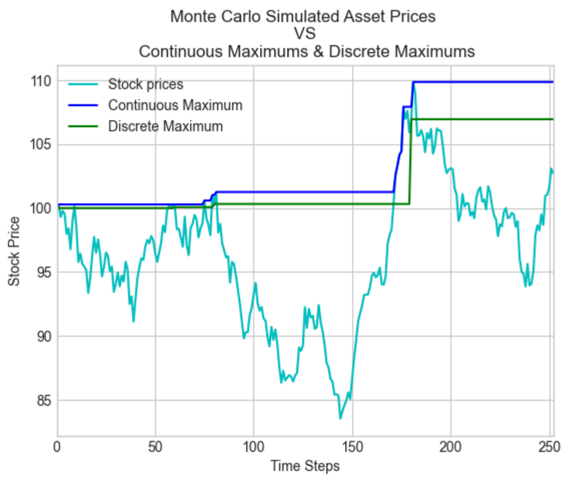


In [ ]:
```
# Discrete Stock maximums
MD = {}

MD = ExoticOptions.discrete_maximum(MD, S_Path, period)

Discrete_max_stock = pd.DataFrame(MD)

for index in range(0, n_new, 1):
    plt.plot(Discrete_max_stock[index])
    plt.grid(True)
    plt.xlabel('Time Steps')
    plt.xlim(0,t)
    plt.ylabel('Stock Price')
    plt.title('Simulated Discrete Maximums');
```



2.4.3 Plot the Maximums vs Stock Price

In [ ]:
```
plt.plot(S_Path[index], color = 'c', label = 'Stock prices')
plt.plot(Continuous_max_stock[index], color = 'b', label = 'Continuous Maximum')
plt.plot(Discrete_max_stock[index], color = 'g', label = 'Discrete Maximum')

plt.grid(True)
plt.xlabel('Time Steps')
plt.xlim(0,t)
plt.ylabel('Stock Price')
plt.title('Monte Carlo Simulated Asset Prices \n VS \n Continuous Maximums & Discrete Maximums')
plt.legend();
```



2.4.4 Lookback Calls - Continuous vs Discrete; Strike vs Rate

In [ ]:
```
# Continuous Lookback Strike Call
clsc = ExoticOptions.payoff_mean(r, T, S_Path, Continuous_max_stock)
# Continuous Lookback Rate Call
clrc = ExoticOptions.payoff_mean(r, T, Continuous_max_stock, E)


# Discrete Lookback Strike Call
dlsc = ExoticOptions.payoff_mean(r, T, S_Path, Discrete_max_stock)
# Discrete Lookback Rate Call
dlrc = ExoticOptions.payoff_mean(r, T, Discrete_max_stock, E)
```

In [ ]:
```
print("Strike Call")

print(tabulate([["Continuous Maximum", clsc],
               ["Discrete Maximum", dlsc]],
               floatfmt=".4f", tablefmt="fancy_grid"))

print("Rate Call")

print(tabulate([["Continuous Maximum", clrc],
               ["Discrete Maximum", dlrc]],
               floatfmt=".4f", tablefmt="fancy_grid"))
```

Strike Call

Continuous Maximum	0.0000
Discrete Maximum	0.6187

Rate Call

Continuous Maximum	10.1700
Discrete Maximum	7.7458

2.4.5 Lookback Puts - Continuous vs Discrete; Strike vs Rate

In [ ]:
```
# Continuous Lookback Strike Put
clsp = ExoticOptions.payoff_mean(r, T, Continuous_max_stock, S_Path)
# Continuous Lookback Rate Put
clrp = ExoticOptions.payoff_mean(r, T, E, Continuous_max_stock)


# Discrete Lookback Strike Put
dlsp = ExoticOptions.payoff_mean(r, T, Discrete_max_stock, S_Path)
# Discrete Lookback Rate Put
dlrp = ExoticOptions.payoff_mean(r, T, E, Discrete_max_stock)
```

In [ ]:
```
print("Strike Put")

print(tabulate([["Continuous Maximum", clsp],
               ["Discrete Maximum", dlsp]],
               floatfmt=".4f", tablefmt="fancy_grid"))

print("Rate Put")

print(tabulate([["Continuous Maximum", clrp],
               ["Discrete Maximum", dlrp]],
               floatfmt=".4f", tablefmt="fancy_grid"))
```

Strike Put

Continuous Maximum	8.3584
Discrete Maximum	6.5530

Rate Put

Continuous Maximum	0.0000
Discrete Maximum	0.0000

3. Varying the data to see the affect on option prices

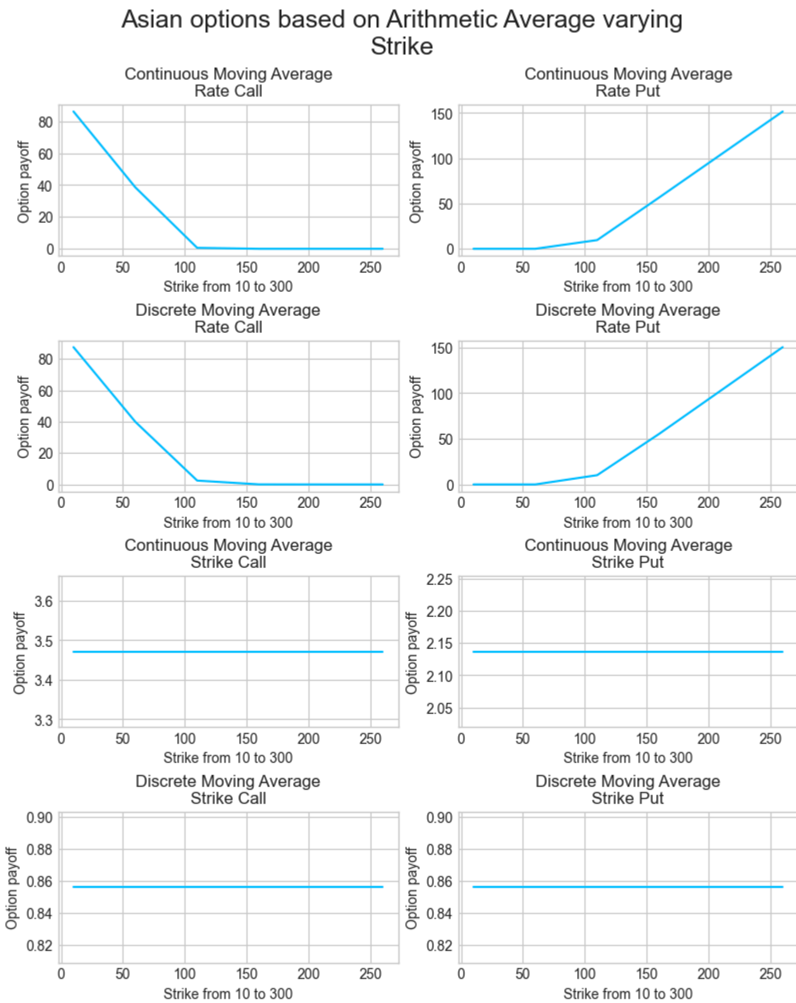
In this session, below variables are changed to see affects on the options payoff results:

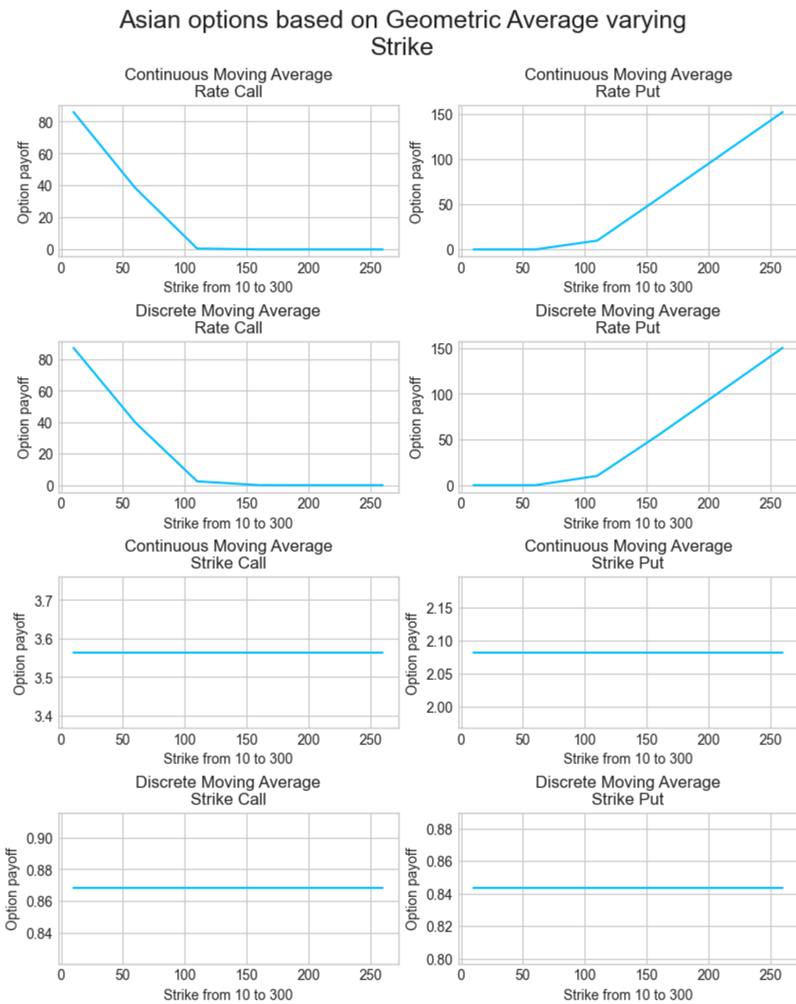
- strike price
- volatility
- risk free rate
- horizon
- timesteps

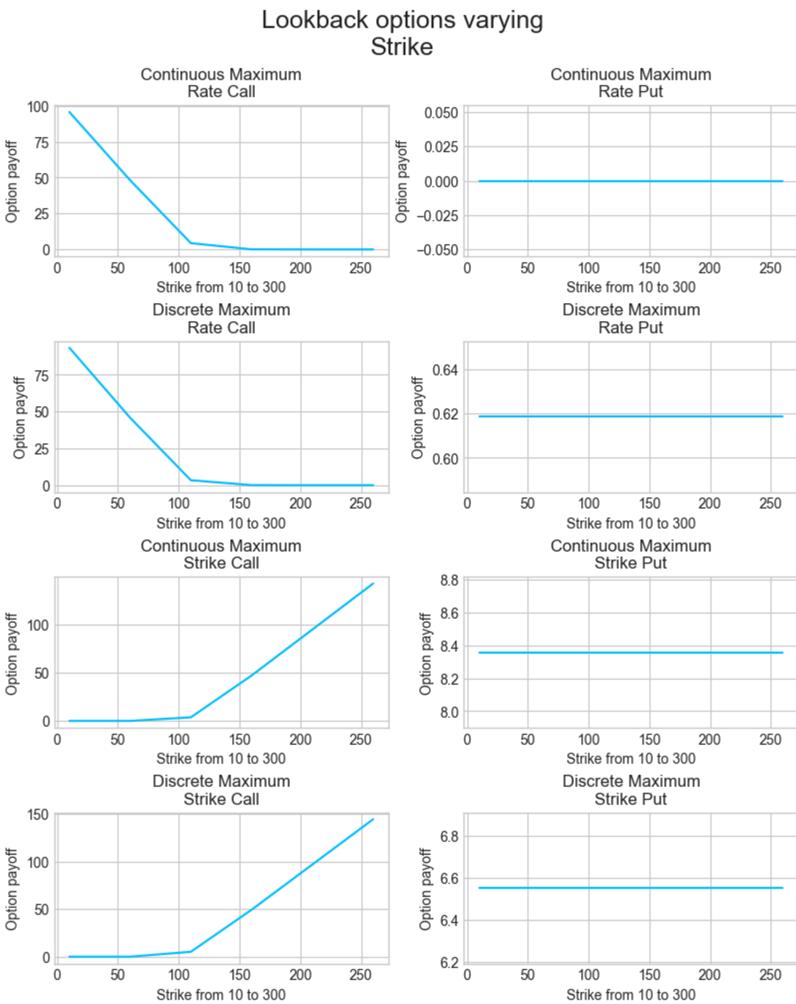
In [ ]:
```
s0 = 100
E = 100
T = 1
vol = 0.2
r = 0.05
t = 252
n_sims = 100
period = 20
```

3.1 Varying the strike price E

In [ ]:
```
ExoticOptions.varying_parameters(s0, r, vol, T, t, n_sims, period, E, 'E', 10, 300, 50)
```




Asian options based on Geometric Average varying Strike
Continuous Moving Average Rate Call
Continuous Moving Average Rate Put
Discrete Moving Average Rate Call
Discrete Moving Average Rate Put
Continuous Moving Average Strike Call
Continuous Moving Average Strike Put
Discrete Moving Average Strike Call
Discrete Moving Average Strike Put
Option payoff
Strike from 10 to 300

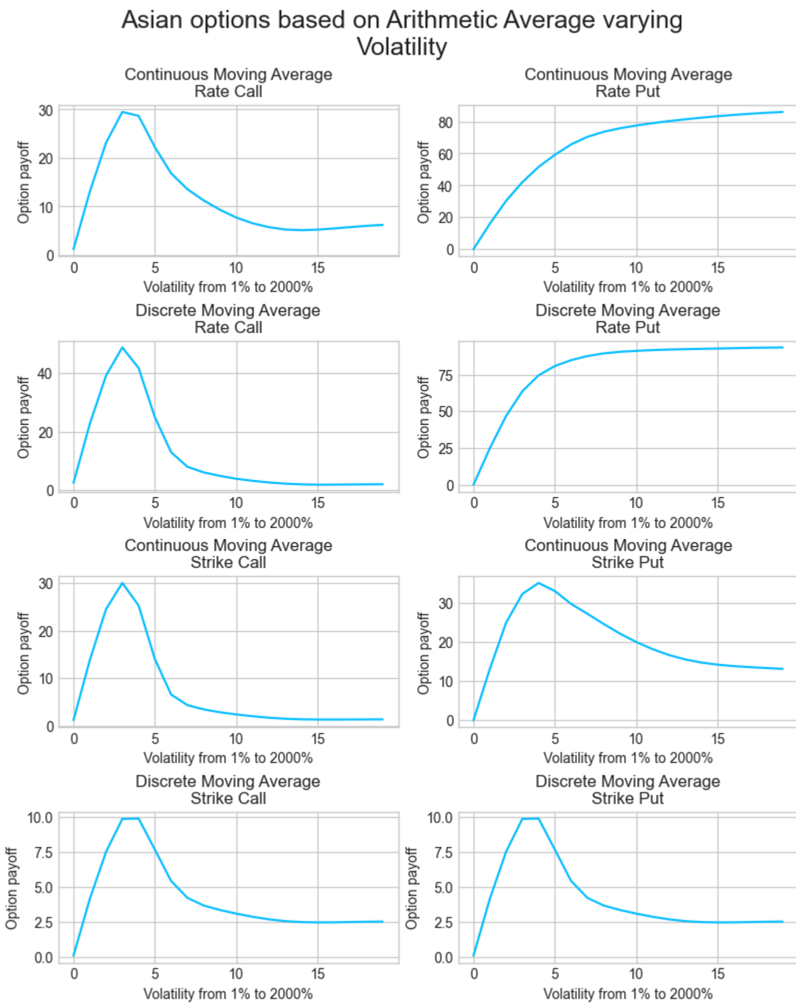


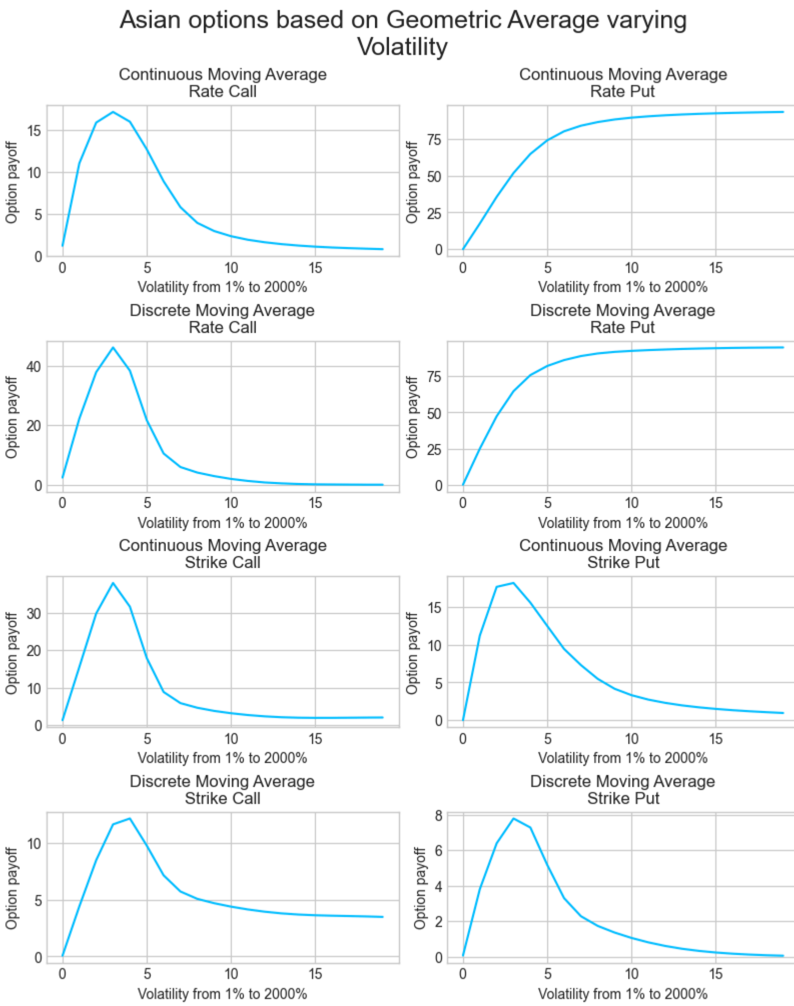
3.1.1 Observations

- For Asian Strike Call & Strike Put Options, their payoffs don't change as the Strike varies.
- For Asian Rate Call Options, as the Strike decreases the Call payoff increases.
- For Asian Rate Put Options, as the Strike increases the Put payoff increases.
- For Lookback Put Options, they don't make changes as the Strike varies.
- For Continuous Maximum Strike Calls & Discrete Maximum Strike Calls: Strike decrease, payoff increase.
- For Continuous Maximum Rate Calls & Discrete Maximum Rate Calls: Strike increase, payoff increase.

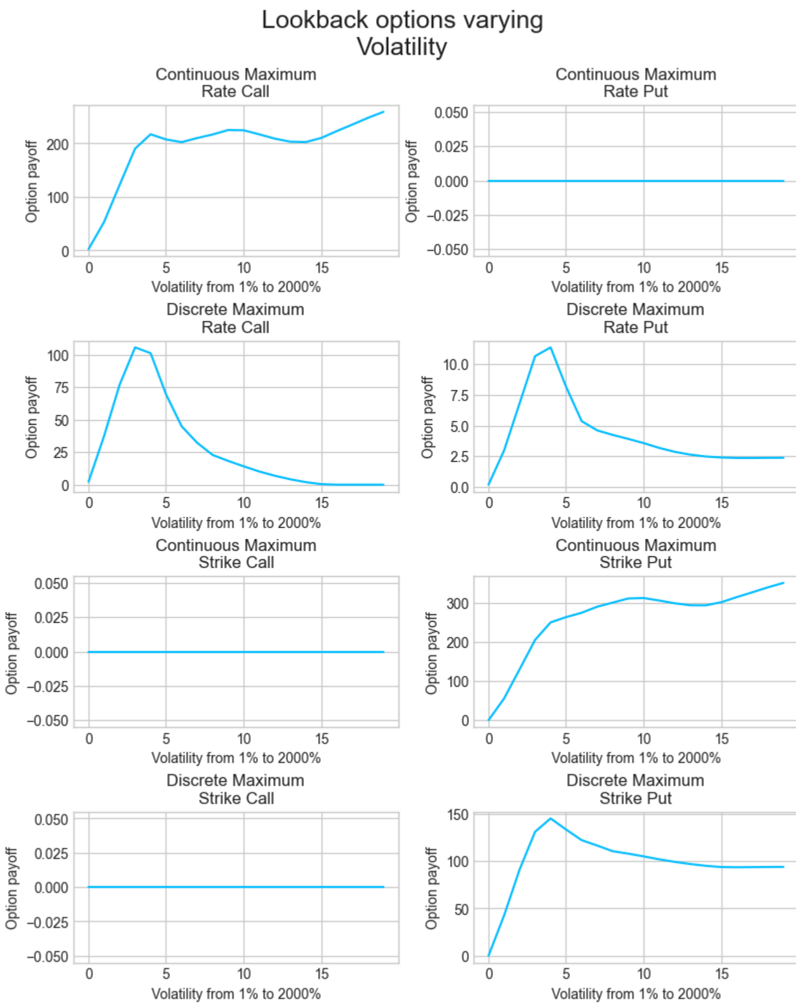
3.2 Varying the Volatility σ

In [ ]:
```
ExoticOptions.varying_parameters(s0, r, vol, T, t, n_sims, period, E, 'vol', 0.01, 20.00, 1)
```


Asian options based on Arithmetic Average varying
Volatility
Continuous Moving Average
Rate Call
Continuous Moving Average
Rate Put
Discrete Moving Average
Rate Call
Discrete Moving Average
Rate Put
Continuous Moving Average
Strike Call
Continuous Moving Average
Strike Put
Discrete Moving Average
Strike Call
Discrete Moving Average
Strike Put
Option payoff
Volatility from 1% to 2000%



Asian options based on Geometric Average varying Volatility
Continuous Moving Average Rate Call
Continuous Moving Average Rate Put
Discrete Moving Average Rate Call
Discrete Moving Average Rate Put
Continuous Moving Average Strike Call
Continuous Moving Average Strike Put
Discrete Moving Average Strike Call
Discrete Moving Average Strike Put
Option payoff
Volatility from 1% to 2000%

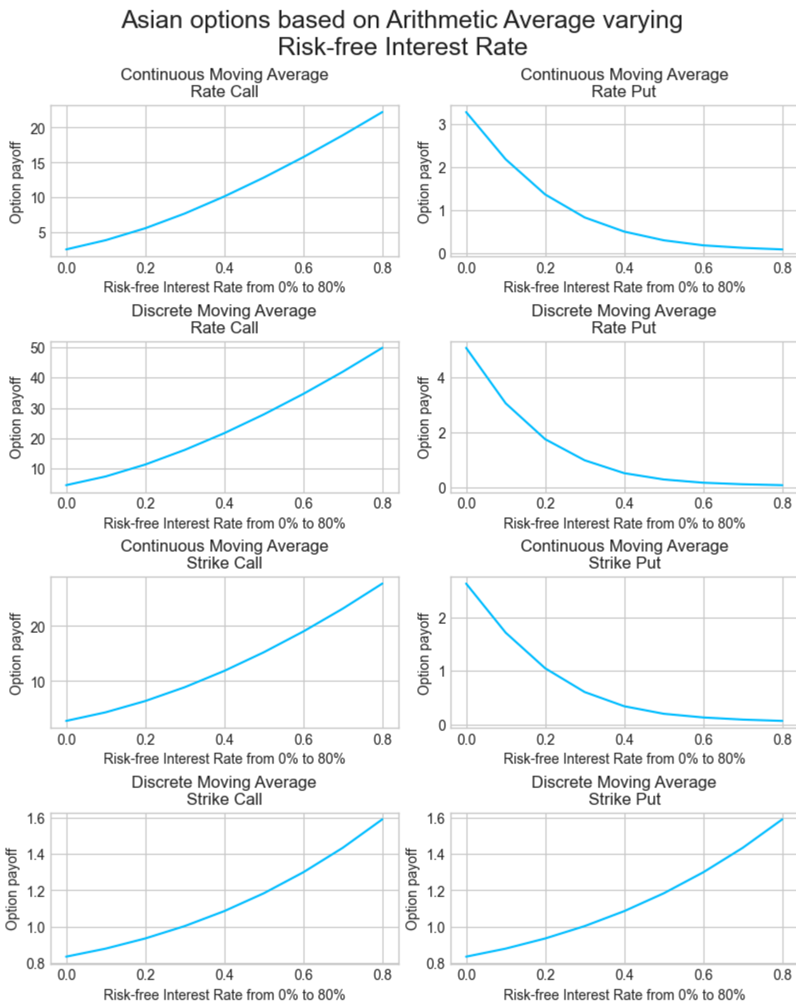


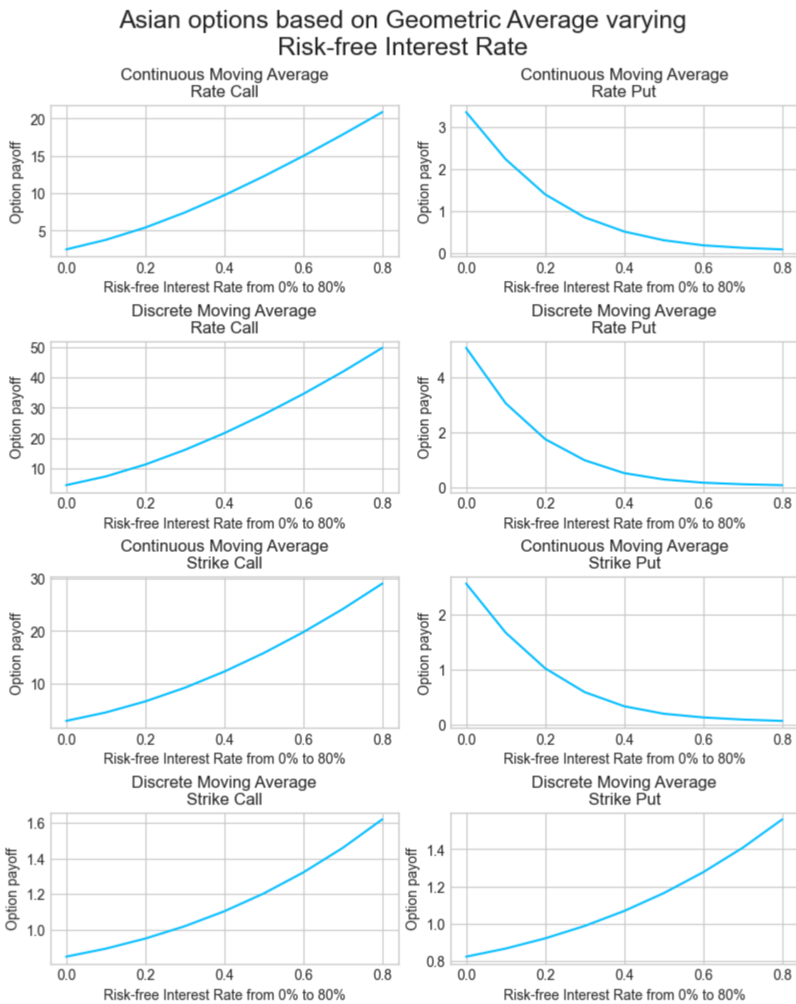
3.2.1 Observations

- For Asian Rate Put Options, their payoffs increase as the volatility increases.
- For the rest Asian Options, they have peaks at around 300%-400%.
- For Lookback Continuous Maximum Rate Put & Lookback Strike Call Options, their payoffs don't change as the volatility varies.
- For the rest Continuous Maximum options, their payoffs increase as the volatility increases.
- For the rest Discrete Maximum options, they have peaks at around 400%.

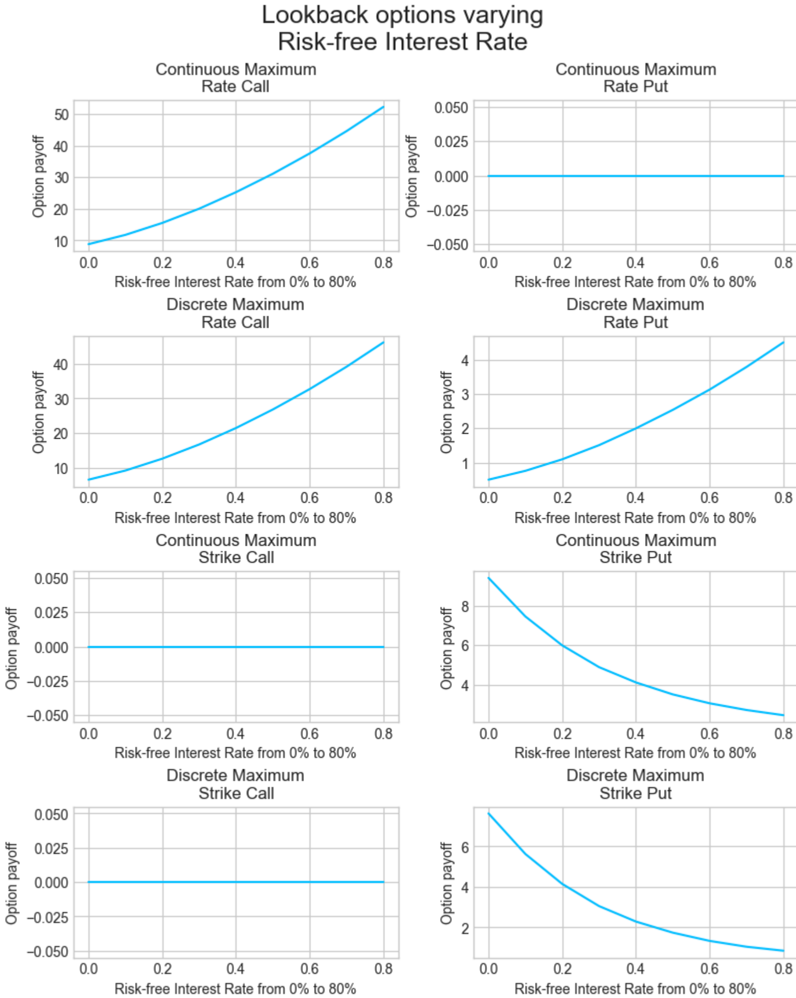
3.3 Varying the risk free interest rate r

In [ ]:
```
ExoticOptions.varying_parameters(s0, r, vol, T, t, n_sims, period, E, 'r', 0, 0.8, 0.1)
```


Asian options based on Arithmetic Average varying Risk-free Interest Rate
Continuous Moving Average Rate Call
Continuous Moving Average Rate Put
Discrete Moving Average Rate Call
Discrete Moving Average Rate Put
Continuous Moving Average Strike Call
Continuous Moving Average Strike Put
Discrete Moving Average Strike Call
Discrete Moving Average Strike Put
Option payoff
Risk-free Interest Rate from 0% to 80%



Asian options based on Geometric Average varying
Risk-free Interest Rate
Continuous Moving Average
Rate Call
Continuous Moving Average
Rate Put
Discrete Moving Average
Rate Call
Discrete Moving Average
Rate Put
Continuous Moving Average
Strike Call
Continuous Moving Average
Strike Put
Discrete Moving Average
Strike Call
Discrete Moving Average
Strike Put
Option payoff
Risk-free Interest Rate from 0% to 80%

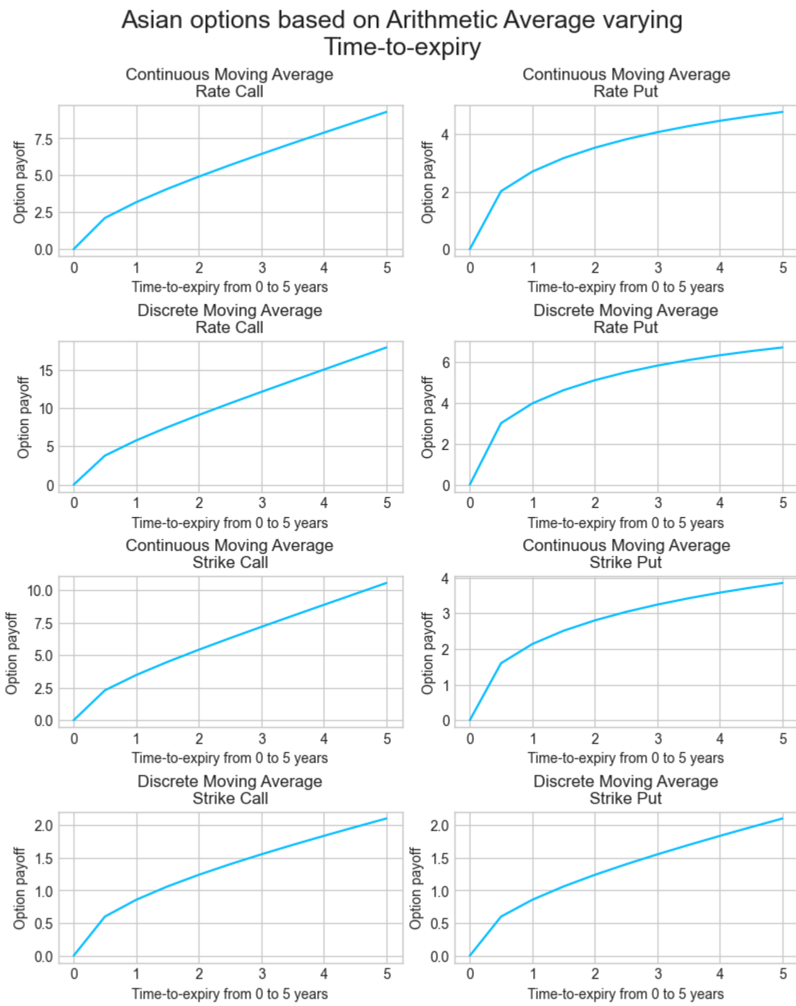


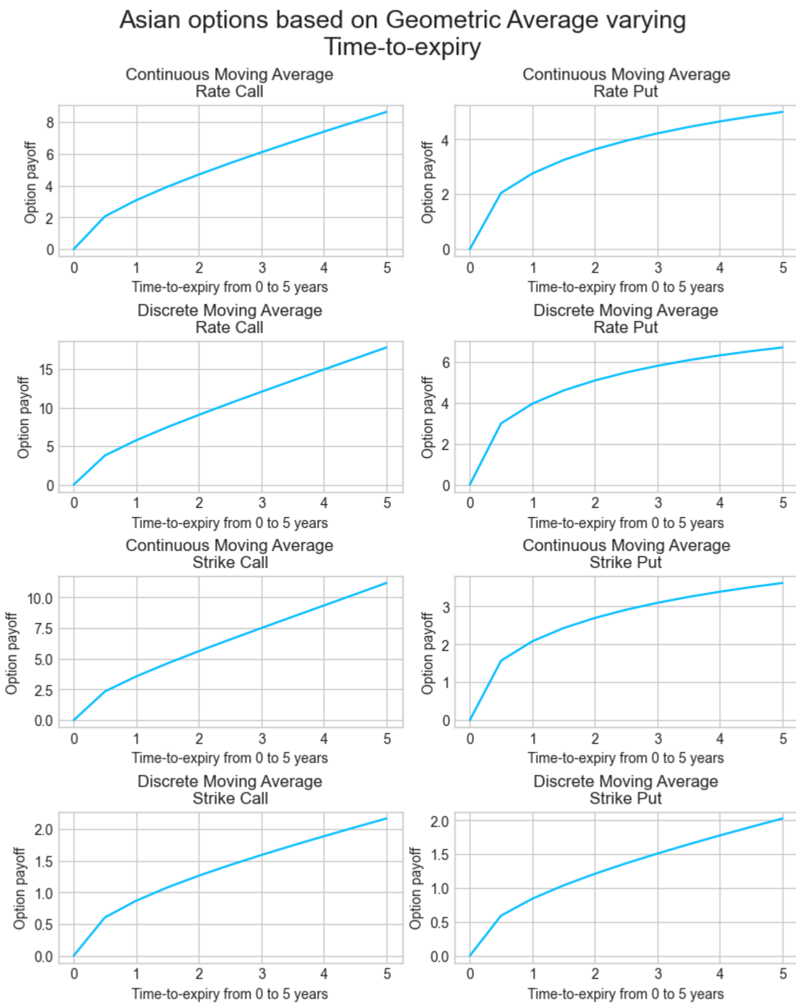
3.3.1 Observations

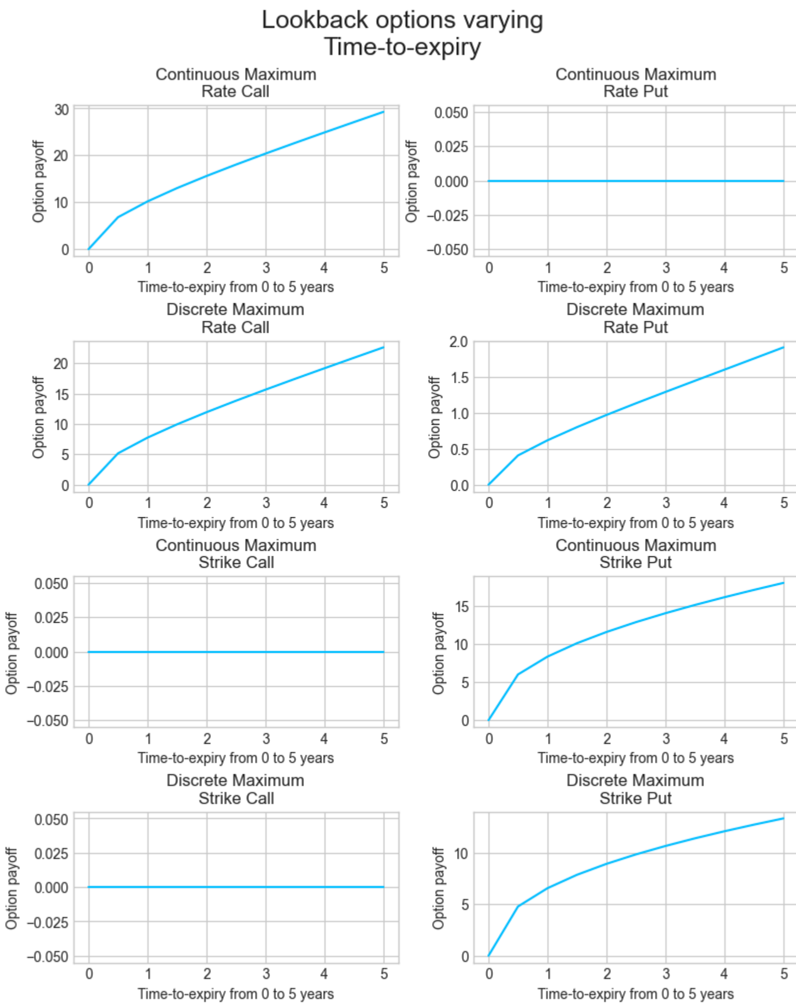
- For Asian Rate Call, Strike Call & Discrete Average Strike Put Options: r decreases, payoff increases.
- For the rest Asian Options: r decreases, payoff decreates.
- For Continuous Maximum Rate Put & Strike Call Lookback Options: their payoffs don't change as the r varies.
- For Maximum Rate Call & Discrete Maximum Rate Put Lookback Options: r decreases, payoff increases.
- For Maximum Strike Put Lookback Options: r decreases, payoff decreases.

3.4 Varying the horizon T

In [ ]:
```
ExoticOptions.varying_parameters(s0, r, vol, T, t, n_sims, period, E, 'T', 0, 5, 0.5)
```


Asian options based on Arithmetic Average varying Time-to-expiry
Continuous Moving Average Rate Call
Continuous Moving Average Rate Put
Discrete Moving Average Rate Call
Discrete Moving Average Rate Put
Continuous Moving Average Strike Call
Continuous Moving Average Strike Put
Discrete Moving Average Strike Call
Discrete Moving Average Strike Put
Option payoff
Time-to-expiry from 0 to 5 years


Asian options based on Geometric Average varying Time-to-expiry
Continuous Moving Average Rate Call
Continuous Moving Average Rate Put
Discrete Moving Average Rate Call
Discrete Moving Average Rate Put
Continuous Moving Average Strike Call
Continuous Moving Average Strike Put
Discrete Moving Average Strike Call
Discrete Moving Average Strike Put
Option payoff
Time-to-expiry from 0 to 5 years



3.4.1 Observations

We are expected to see that the longer time to expiry, higher option prices. The reason why we are seeing flat lines for 3 of them was because the Strike value defined.

4. Conclusion

With this project, we can see how to price exotic options such as Asian and lookback options using the Euler-Maruyama scheme and how the payoff of them may vary depending on the averaging methods and other parameters. To specify, we manage to include Asian call option with fixed delivery price, Asian put option with fixed delivery price, Asian call option with floating delivery price, Asian put option with floating delivery price, Lookback call option with fixed delivery price, Lookback put option with fixed delivery price, Lookback call option with floating delivery price, Lookback put option with floating delivery price all into the simulations. As for the parameters, we are able to run assuptions on parameters such as:

- strike price
- volatility
- risk free rate
- horizon
- timesteps

5. References

- CQF Program, Python Primers
- CQF Program, Python Lab: Monte Carlo Simulation.
- CQF Program, Tutorial: Numerical Methods & Further Topics in Monte Carlo.
- Boyle, P. & Potapchik, A. (2008), 'Prices and sensitivities of asian options: A survey', Insurance: Mathematics and Economics 42(1), 189–211
- Paul Wilmott, Paul Wilmott on Quantitative Finance Vol 3
- Mark Joshi, Quant Job Interview Questions and Answers