Instructions

- You will submit your solution in a **single file**, which contains the classes `AsianOption`, `Asset`, `BlackScholesAsset`, and `Option`.
- You may **only** import (from) the `math`, `matplotlib.pyplot`, `numpy`, `random`, and `statistics` modules.
- You do not need to include code to **demonstrate** these classes, except for your own testing purposes (see Testing your own code section below).
- Your code should obey the PEP 8 and numpy docstring formats. Instructions for setting up style checking in Spyder can be found here.

- Your file should be written entirely in *English*, including comments and docstrings.
- **You are not allowed to plagiarise.** This means that you are not allowed to directly copy the code from any online sources or any of your fellow students. Direct group work is also not allowed. What this means is that while you are allowed (and even encouraged) to discuss the contents of this module with your fellow students, each of you must write your own code. See this Canvas page on Academic Integrity and Plagiarism for more information.
- If you need to request an **extension**, please contact the School of Mathematics Wellbeing Officer: mathswelfare@contacts.bham.ac.uk.

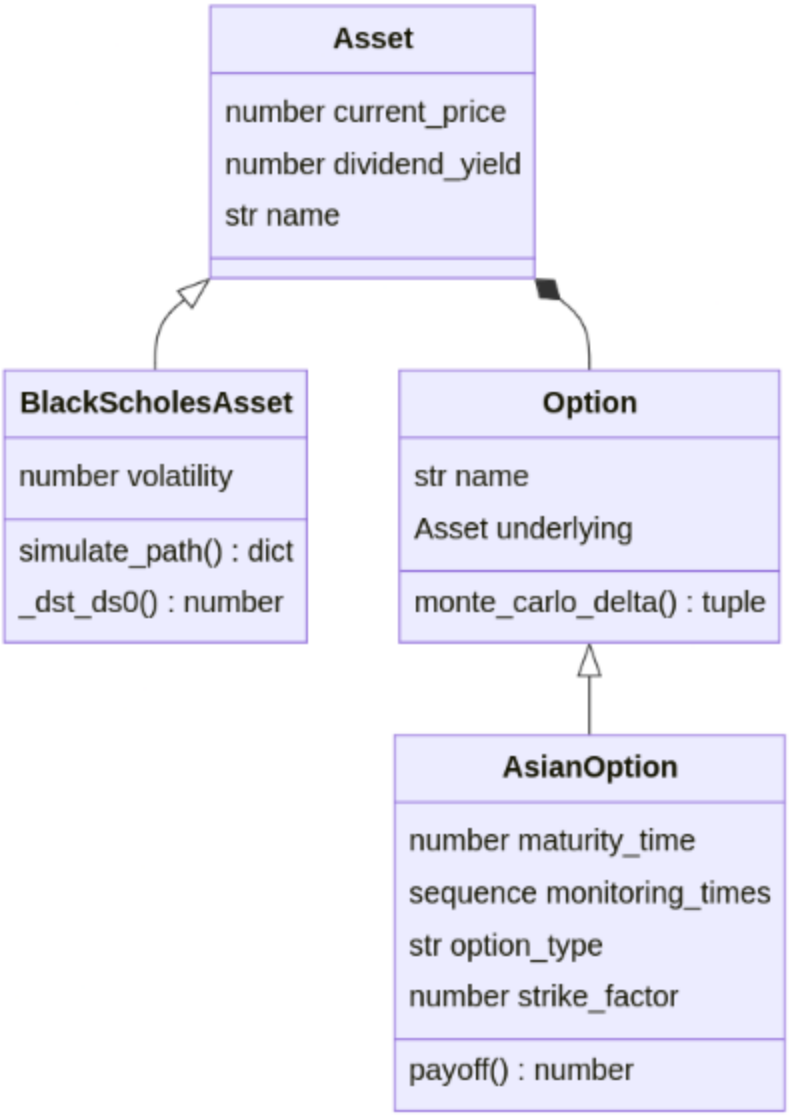
Project description

You will create several classes for Asian option Greek calculations. All time periods are assumed to be measured in years and interest rates, dividend yields, and volatilities are stored as numbers, not percentages. You should verify that parameters satisfy the assumptions in this project (raising your own errors when they are not met), including the following:

- names should be non-empty strings,
- interest rates, prices, times, and volatilities should be positive numbers,
- dividend yields should be non-negative numbers,
- paths are dictionaries with non-negative keys and positive values.

The following class diagram shows the structure your code should have. Each box represents a class, consisting of 3 smaller boxes:

- the top box contains the name of the class,
- the middle box contains the attributes and properties of the class, and
- the bottom box contains the methods of the class.


Asset
number current_price
number dividend_yield
str name
BlackScholesAsset
number volatility
simulate_path() : dict
_dst_ds0() : number
Option
str name
Asset underlying
monte_carlo_delta() : tuple
AsianOption
number maturity_time
sequence monitoring_times
str option_type
number strike_factor
payoff() : number

An arrow represents an inheritance relationship ("is a") whilst a diamond represents a composition relationship ("has a").

`Asset` class

This is the base class for assets in this project, and is intended to be used for common behaviour of all `Asset` classes (even though there is only one subclass in this project). `Asset`s should have the following attributes:

- `name`: the name of the asset;
- `current_price`: the current price of the asset;
- `dividend_yield`: the (continuous) dividend yield of the asset.

The constructor for this class should take `name` and `current_price` as positional arguments (in order), with `dividend_yield` as a keyword-only parameter with default value 0.

`BlackScholesAsset` class

This class is a subclass of `Asset` and assumes that the asset prices $(S(\tau))_{\tau\in[0,\infty)}$ follow a geometric Brownian motion model. Recall that for each $\tau \in [0, \infty)$,

$$S(\tau) = S(0)\exp\left(\left(r - d - \frac{\sigma^2}{2}\right)\tau + \sigma W(\tau)\right)$$

where

- r is the risk-free interest rate,
- d is the dividend yield,
- σ is the volatility, and
- $(W(\tau))_{\tau\in[0,\infty)}$ is a Wiener process.

The constructor for this class should have an additional `volatility` attribute, which the constructor should have as a keyword-only parameter.

This class should contain a `simulate_path` method to simulate asset. This method takes the following parameters:

- `simulated_times`: a strictly-increasing, non-empty sequence of positive times to be simulated;
- `interest_rate`: the risk-free interest rate;
- `current_price`: this optional parameter has default value `None`, if instead a number is provided, then this value should be used for the start of the simulation (i.e. used as $S(0)$) instead of the `current_price` attribute.

The last two parameters should be keyword-only. If the simulated times are $(t_1, \ldots, t_n)$ and $S(t)$ is the (random variable) price of the asset at time t (where $S(0)$ represents the current price), then this method should return a dictionary with keys $t_0 := 0, t_1, t_2, \ldots, t_n$ with corresponding values $S(t_0), \ldots, S(t_n)$. To simulate from $S(t_{i+1})$ from $S(t_i)$, you should use the geometric Brownian motion model presented above and the properties of the Wiener process.

This class should also have a helper function, `_dst_ds0`, which computes the derivative $\frac{dS(t)}{dS(0)}$ and will be used in the Monte-Carlo delta calculation.

This derivative should be calculated using the realised asset prices in `path`, and this method should have the following parameters:

- `path` is a dictionary representing a simulated path of asset prices, including the initial price $S(0)$ to be used.
- `time` is the time to calculate the derivative at (i.e. t). This should be a key in the `path`, i.e. $S(t)$ has already been realised.

`Option` class

This class is the base class for all options. The constructor for this class takes the following parameters:

- `name`: the name of the option,
- `underlying`: the underlying `Asset` of the option.

This class also contains a `monte_carlo_delta` method, which will calculate the Δ of this option using a Monte-Carlo method. This method should have the following parameters, with the last 2 being keyword-only:

- `simulations`, the (whole) number of simulations to perform (at least 2).
- `confidence_level`: the confidence level for the confidence interval, which should be a number strictly between 0 and 1, with default value 0.95.
- `interest_rate`: the risk-free interest rate.

Using the `simulate_path` method, simulate a path of option prices at the monitoring times and calculate Δ for this path (see below). After performing this procedure `simulations`-many times, you should then calculate the mean $\overline{\Delta}$ and confidence interval

$$(\Delta_-, \Delta^+) = \left(\overline{\Delta} - \frac{s_\Delta}{\sqrt{n}}\Phi^{-1}\left(\frac{1+\gamma}{2}\right), \overline{\Delta} + \frac{s_\Delta}{\sqrt{n}}\Phi^{-1}\left(\frac{1+\gamma}{2}\right)\right),$$

where

- s_Δ is the sample standard deviation,
- n is the number of simulations performed,
- γ is the confidence level,
- Φ is the cumulative distribution function of a standard normal variable.

This method should finally return a triple $\left(\Delta_-, \overline{\Delta}, \Delta_+\right)$. You may find the `NormalDist` class from the `statistics` module helpful.

As the calculation for an estimate for Δ depends on the type of option (in particular on its payoff function), you should implement a `_path_delta` method in subclasses to calculate Δ for each path (see next section).

`AsianOption` class

This class is for Asian options with floating strike, arithmetic averaging, and a finite set of monitoring times. In addition to the parameters for the `Option` class, the constructor should take the following additional, keyword-only parameters:

- `option_type`: the type of option, either `'call'` or `'put'`;
- `monitoring_times`: a strictly-increasing, non-empty sequence of positive times which are used in the averaging calculation for payoffs;
- `strike_factor`: a positive number used in payoff calculations.

This class should also have a `maturity_time` property, which is the last exercise time of the option.

The `payoff` method returns the payoff of the option, using the following formulas, where k is the `strike_factor`, $S(t)$ is the price of the asset and $A = \frac{1}{n}\sum_{i=1}^{n} S(t_i)$, where $t_1 < t_2 < \cdots < t_n$ are the monitoring times of the option:

- $\max(S(t_n) - kA, 0)$ if `option_type` is `'call'`.
- $\max(kA - S(t_n), 0)$ if `option_type` is `'put'`.

This method takes a `path` dictionary parameter, representing a simulated path of underlying asset prices. You should verify that it contains all `monitoring_times` as keys.

The `_path_delta` method should take the following parameters:

- `path` is a dictionary representing a simulated path of asset prices, including the initial price $S(0)$ to be used;
- `interest_rate`: the risk-free interest rate.

This method should calculate the derivative $\frac{\mathrm{d}V}{\mathrm{d}S(0)}$, where V is the option value (i.e. *discounted* expected payoff), using the `_dst_ds0` method to calculate the derivatives $\frac{\mathrm{d}S(t)}{\mathrm{d}S(0)}$ for each monitoring time $t_1, \ldots, t_n$. This derivative should be calculated using the realised asset prices in `path`, and you should verify that the path contains these monitoring times.

Testing your own code

As I will be importing your code as a module, to help aid my marking please put all your test code (i.e. creating instances of classes, running methods, etc.) inside the following code block, which should be at the end of your file. This will ensure that this code is only run if the file is run and not if it is imported:

```
if __name__ == '__main__':
    # YOUR TEST CODE HERE
```

As you are generating random variables in these methods, you may find it helpful to test these methods by running a large number of simulations and calculating a confidence interval for the means, using the definitions of $S(t)$ in the Black-Scholes model to determine the actual means these methods

As you are generating random variables in these methods, you may find it helpful to test these methods by running a large number of simulations and calculating a confidence interval for the means, using the definitions of $S(t)$ in the Black-Scholes model to determine the actual means these methods should produce. Note that $\mathrm{Lognormal}(a, b^2) = \exp(\mathcal{N}(a, b^2))$ has mean $\exp\left(a + \frac{b^2}{2}\right)$.