Problem Set 2
Monte Carlo Methods for Option Pricing

- This homework consists of three questions, each carrying equal marks. Your code will be graded by a Python script which compares your result against the baseline result. It is therefore important that your Python stack is identical to the baseline so that you avoid losing marks even though your code is correct.
- Submit a 7z compressed archive containing three modules, `BSMonteCarlo.py`, `MCStockPrices.py`, `MCOptionPrices.py` and a report in PDF format.
- Be sure to include your name as part of the archive filename, and at the top of your report.

Problem 1. *Write a function to price vanilla european-exercise options*

```
def BSMonteCarlo( S0, K, T, sigma, checkpoints,  rateCurve,
    samples=None )
```

where

- $S0, K, T,$ *sigma are the underlying price, the option strike, the maturity and the volatility respectively.*
- '`checkpoints`' *is an ordered list of integer sample counts at which to return the running mean, standard deviation, and estimated error.*
- '`rateCurve`' *is an* `InterestRateCurve` *stored as a numpy array.*
- '`samples`' *is a numpy array of uniform random samples to use. If this input is* `None` *then your code should generate a fresh 1-dimensional sample array with* M *elements where* M *is the final entry in* `checkpoints`*.*

Note that `samples` *(when specified) must be a list with at least as many elements as the last entry in* `checkpoints`*. Raise an exception if this is not the case.*

This function must return a dictionary with the following entries:

```
{ 'TV':   , # The final value ( i.e. mean at checkpoints[-1] )
  'Means':   , # The running mean at each checkpoint
  'StdDevs':   , # The running standard deviation at each checkpoint
  'StdErrs':   , # The running standard error at each checkpoint
}
```

Problem 2. *Given uniform samples we can generate* M *paths of stock prices at given times* `t` *under the assumption of geometric brownian motion. Write a function*

```
def MCStockPrices( S0, sigma, rateCurve, t,
     samples, integrator):
```

where

- `'S0'` *is the stock prices at time* t_0.
- `'sigma'` *is the constant volatility.*
- `'rateCurve'` *is an* `InterestRateCurve` *stored as a numpy array.*
- `'t'` *is an* `array` *of fixing times* $t_i, i = 1 \ldots N$ *to simulate to.*
- `'samples'` *is an* `array` *of uniform random samples to use. The length of* `samples` *should be* $N \times M$ *where* N *is the number of fixing times and* M *is the number of paths.*
- `integrator` *controls how the samples are generated according to the following value list*
  - `'standard'`*, where the paths are generated by using the solution of the Black-Scholes SDE step-by-step*
  - `'euler'`*, to use Euler-method integration of the Black-Scholes SDE*
  - `'milstein'`*, to use Milstein-method integration of the Black-Scholes SDE*

The function should return a `numpy array` *of simulated stock prices having the same dimensions as* `samples`.

Problem 3. *Combine your* `BSMonteCarlo` *and* `MCStockPrices` *functions from Questions 1 and 2 to write a function*

```
def MCOptionPrices( S0, K, T, rateCurve, sigma, t,
     checkpoints, samples, integrator):
```

to approximate the prices of European Options where

- `'t'` *is an* `array` *of fixing times* $t_i, i = 1 \ldots N$ *to simulate to.*
- `'K'` *is the strike price.*
- `'T'` *is the expiration date of the European option.*
- `'rateCurve'` *is an* `InterestRateCurve` *stored as a numpy array.*
- `'checkpoints'` *is an ordered list of integer sample counts in the range* $[1, M]$ *at which to return the running mean, standard deviation, and estimated error.*
- `'samples'` *is an* `array` *of uniform random samples to use. The length of* `samples` *should be* $N \times M$ *where* N *is the number of fixing times and* M *is the number of paths.*
- `integrator` *controls how the samples are generated according to the following value list*
  - `'standard'`*, where the paths are generated by using the solution of the Black-Scholes SDE step-by-step*
  - `'euler'`*, to use Euler-method integration of the Black-Scholes SDE*
  - `'milstein'`*, to use Milstein-method integration of the Black-Scholes SDE*

This function must return a dictionary with the following entries:

```
{ 'TV':  , # The final value ( i.e. mean of option price at time t_0
               using NxM uniform random samples)
  'Means':  , # The running mean at each checkpoint
  'StdDevs':  , # The running standard deviation at each checkpoint
  'StdErrs':  , # The running standard error at each checkpoint
}
```

Submit a brief report comparing the error properties of each integrator with the exact option price. Your studies should compare the convergence properties of each scheme as M and N vary.