General Instructions

1. Question 4 is based on `MClib` and the particular implementation also uses `DOPlib`. The documentation to these libraries is also available on the course website.

2. Each question is set in its own project: `SampleExam2015MTQ1`, `SampleExam2015MTQ2`, `SampleExam2015MTQ3` and `SampleExam2015MTQ4`. Download, unzip and add these projects to `[CppCourse]\[SampleExam2015]` folder as shown below:

```
[CppCourse]
   -> [boostRoot]
   ...
   -> [SampleExam2015]
      -> [SampleExam2015MTQ1]
         -> CMakeLists.txt
         -> MyRand.hpp
         -> SampleExam2015Q1Exercises.hpp
         -> [Src]
            -> ...
      -> [SampleExam2015MTQ2]
         -> ...
      -> [SampleExam2015MTQ3]
         -> ...
      -> [SampleExam2015MTQ4]
         -> ...
   ...
```

3. Open the text file `[CppCourse]\CMakeLists.txt`, uncomment the following lines by removing the `#`:

```
#add_subdirectory(SampleExam2015/SampleExam2015MTQ1)
#add_subdirectory(SampleExam2015/SampleExam2015MTQ2)
#add_subdirectory(SampleExam2015/SampleExam2015MTQ3)
#add_subdirectory(SampleExam2015/SampleExam2015MTQ4)
```

   and save the file. This registers the project with `cmake`.

4. Run `cmake` in order ot generate the project.

5. Each project is to be compiled separately. Each project, if compiles and runs properly, will create a text file, `SampleExamQ1_output.txt`, `SampleExamQ2_output.txt` etc.

Question 1

The exercise is about the Monte Carlo pricing of options on bonds. All formulas are given, no interest-rate specific background knowledge is required.

The exercise is based on the Vasicek short rate model that derives the bond price dynamics from the instantaneous rate r_t. The model has three parameters: κ (speed of mean reversion), ϕ (mean reversion leve), σ (volatility of r_t). The functions in this exercise are expressed in terms of these parameters and the instantaneous rate.

The header file `SampleExam2015Q1Exercises.hpp` contains the declaration of four functions. These functions are to be implemented in `cpp` file(s) that you will create.

The header file also contains some type definitions that are used throughout this project.

You will need to generate random variables from the standard normal distribution. The `MyRand.hpp` header provides the following functions.

```
double NormalDist();
void NormalDist(std::vector<double> & vArg);
```

The first one returns one realisation of a standard normal variable, whereas the second takes a vector by reference, and fills it up with standard normal variables.

Question 1a

The first exercise is to implement the bond price (the time-t value of 1 currency unit paid at T with $T \geq t$) formula:

$$P(r_t, t, T, \kappa, \phi, \sigma) = A(t, T, \kappa, \phi, \sigma)e^{-B(t,T,\kappa)r_t},$$

where

$$B(t, T, \kappa) = \frac{1}{\kappa}\left[1 - e^{-\kappa(T-t)}\right],$$

$$A(t, T, \kappa, \phi, \sigma) = \exp\left\{\left(\phi - \frac{\sigma^2}{2\kappa^2}\right)[B(t, T, \kappa) - T + t] - \frac{\sigma^2}{4\kappa}B(t, T, \kappa)^2\right\}.$$

The declaration of the function is:

```
1 double bondPrice(double r, double t, double T, double kappa, double phi, double
      sigma);
```

where input arguments are specified as below:

- `r` instantaneous rate r_t,
- `t` time t,
- `T` maturity T of bond,
- `kappa` mean reversion speed κ,
- `phi` mean reversion level ϕ,
- `sigma` volatility of instantaneous rate σ.

Question 1b

In the Vasicek model the instantaneous rate r_t is normally distributed with mean and variance:

$$\mathbb{E}[r_t|r_0] = r_0 e^{-\kappa t} + \phi\left(1 - e^{-\kappa t}\right) \tag{1}$$

$$\text{StDev}[r_t|r_0] = \sqrt{\frac{\sigma^2}{2\kappa}\left(1 - e^{-2\kappa t}\right)}. \tag{2}$$

The following function is to implement the expected value (1).

```
double expectedValueOfR(double r0,double t, double kappa, double phi);
```

where input arguments are specified as below.

- `r0` initial value of r,
- `t` time,
- `kappa` mean reversion speed κ,
- `phi` mean reversion level ϕ.

The following function is to implement the standard deviation (2).

```
double standardDeviationOfR(double t, double kappa, double sigma);
```

Question 1c

In this exercise you are expected to implement a Monte-Carlo estimation of the value of options on bonds. In particular, you are expected to estimate the time-0 value of the contract that pays off at time T, and the payoff depends on the bond price $P(T, S)$ where $T < S$.

The function to be written is declared as follows.

```
MCResults MonteCarloBond(double r0,
          double kappa,
          double phi,
          double sigma,
          double maturityOfBond,
          double maturityOfOption,
          unsigned long int sampleSize,
          Payoff payoff);
```

where the arguments are specified below.

- `r0` instantaneous rate r_0 at time 0,
- `kappa` mean reversion speed κ,
- `phi` mean reversion level ϕ,

- `sigma` volatility of instantaneous rate σ,
- `maturityOfBond` maturity of the bond S,
- `maturityOfOption` maturity of the option T,
- `sampleSize` number of sample items to generate N,
- `payoff` the payoff function f (as a function of $P(T, S)$).

The bond price at T is a function of r_T. Moreover, r_T is normally distributed (see the previous exercise). Therefore, given the input parameters, the function is to generate a sample of r_T; for each sample item r_T, compute the bond price, and substitute it into the payoff function. Since the payoff is payed at T, it is to be discounted by $P(0, T)$. In particular, the Monte-Carlo estimate is:

$$P(0,T)\frac{1}{N}\sum_{i=1}^{N} f(P(T,S,r_T^{(i)})). \tag{3}$$

The function is to return an instance of `MCResults` that is declared as a `struct` (that is `class` where each member is `public` unless declared otherwise), that has two data members:

- `mean` the Monte-Carlo estimate of the option value,
- `stdev_of_mc_estimate` the estimated standard deviation of the Monte-Carlo estimate.

Question 2

This exercise is about implementing a polynomial interpolator in terms of a function object (a class with an `operator()` member).

The interpolator is constructed from two vectors $X = [x_1, \ldots, x_n]$ and $Y = [y_1, \ldots, y_n]$, and its function operator evaluates the following function $p(x) := p_n(x)$ where p_n is defined recursively as follows.

$$\begin{aligned} p_0(x) &= 0, \\ p_1(x) &= y_1 \\ &\vdots \\ p_{i+1}(x) &= p_i(x) + \underbrace{(y_{i+1} - p_i(x_{i+1}))}_{v_{i+1}:=} \underbrace{\left(\prod_{j=1}^{i}(x - x_j)\right)}_{w_{i+1}(x):=} \underbrace{\frac{1}{\prod_{j=1}^{i}(x_{i+1} - x_j)}}_{z_{i+1}:=} \end{aligned} \tag{4}$$

Note 1 When X and Y are empty vectors, the interpolator implements the constant 0 function.

Note 2 When $n > 0$, the function p is a degree-$(n-1)$ polynomial, and for i=1, ... n, we have $p(x_i) = y_i$.

Note 3 This particular constructions enables the extension of the structure. That is given an interpolator constructed from $X = [x_1, \ldots, x_n]$ and $Y = [y_1, \ldots, y_n]$, and given two real numbers x_{n+1}, and y_{n+1}, the interpolator can be modified at little cost to implement a degree-n polynomial, for which for i=1, ..., n+1, we have $p(x_i) = y_i$ holds.

The header file `SampleExam2015Q2Exercises.hpp` contains the declaration of the class `polynomialInterpolator` that is to implement all these features. Create a `cpp` file, and implement the member functions (as specified below) in it.

Question 2a

The constructor of the class is declared as follows.

```
1    polynomialInterpolator(const dVector & xVec, const dVector & yVec);
```

It takes the following arguments.

- `xVec` the X vector,
- `yVec` the Y vector.

The constructor is to compute the $V = [v_1, \ldots, v_n]$ and $Z = [z_1, \ldots, z_n]$ vectors (as defined in the recursion (4) above – note $v_1 = y_1$ and $z_1 = 1$), and save them in the `m_v` and `m_z`. Moreover, X is to be saved into `m_x`.

Use assertion to check if X and Y have the same size.

Note You may consider implementing the constructor in terms of `addPoint()` (see part 2b).

Question 2b

The `addPoint()` function is declared as below.

```
1    void addPoint(double x, double y);
```

It takes x_{n+1} and y_{n+1}, it is to compute v_{n+1} and z_{n+1} as defined in (4), and append the members `m_x`, `m_v` and `m_z` by x_{n+1}, v_{n+1} and z_{n+1} respectively.

Note that z_{n+1} is not well defined if $x_{n+1} = x_j$ for some $i \leq n$. Use assertions to detect when this occurs.

Question 2c

The polynomial evaluation is to be implemented by the `operator()` member of the class. This is declared as below.

```
1    double operator()(double x) const;
```

The function takes x as argument, and is to evaluate $p(x) = p_n(x)$ via the recursion (4). You may consider evaluating $w_i(x)$ recursively and simultaneously with $p_i(x)$.

Question 3

The header file `SampleExam2015Q3Exercises.hpp` contains the following function declarations together with some type definitions.

```
1  void tridiSolver(const DVector & a, const DVector & b, const DVector & c, DVector
       & x);
```

```
1  DMatrix implicitFiniteDifferenceSolver(double dSmax,
2                        double dT,
3                        double dR,
4                        double dSigma,
5                        size_t n,
6                        size_t m,
7                        PayoffFunction payoff);
```

Question 3a - `tridiSolver`

The function `tridiSolver` solves the following tridiagonal linear system:

$$\begin{pmatrix} b_0 & c_0 & 0 & \cdots & 0 \\ a_1 & b_1 & c_1 & & \\ & \ddots & \ddots & \ddots & \\ & & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & \cdots & 0 & a_n & b_n \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} d_0 \\ d_1 \\ \vdots \\ d_{n-1} \\ d_n \end{pmatrix}$$

The exercise is to implement the Thomas algorithm consists of two parts.

1. Forward sweep:

$$v_i := \begin{cases} \frac{c_i}{b_i} & i = 0 \\ \frac{c_i}{b_i - v_{i-1}a_i} & i = 1, 2, \ldots, n \end{cases}$$

and

$$x_i = \begin{cases} \frac{x_i}{b_i} & i = 0 \\ \frac{x_i - x_{i-1}a_i}{b_i - v_{i-1}a_i} & i = 1, 2, \ldots, n \end{cases}$$

2. Backward substitution

$$x_i = x_i - x_{i+1}v_i \quad i = n-1, n-2, \ldots, 0$$

The arguments of the function:

- `a` diagonal below the main diagonal (first entry not used),
- `b` main diagonal,
- `c` diagonal above the main diagonal (last entry not used),
- `x` the vector $d = (d_0, \ldots, d_n)^T$ on the right hand side of the linear equation; the solution $x = (x_0, \ldots, x_n)$ is to be written into the same vector.

Question 3b - `impicitFiniteDifferenceSolver`

The function `impicitFiniteDifferenceSolver` implements the implicit finite difference method for the Black-Scholes model with n equal spacial steps and m equal time-steps. The tridiagonal matrix of the linear system is determined by

$$\begin{aligned} a_i &= -\tfrac{1}{2}(\sigma^2 i - r) i \Delta t, \text{ for } i = 0, \ldots, n-1 \\ b_i &= 1 + \Delta t(\sigma^2 i^2 + r), \text{ for } i = 0, \ldots, n-1 \\ c_i &= -\tfrac{1}{2}(\sigma^2 i + r) i \Delta t, \text{ for } i = 0, \ldots, n \end{aligned} \tag{5}$$

and $a_n = 0$, $b_n = 1 + r\Delta t$.

The arguments of the function:

- `dSmax` upper end of spacial grid,
- `dT` time to maturity,
- `dR` risk-free interest rate r,
- `dSigma` volatility σ,
- `n` number of spacial grid points,
- `m` number of time-steps,
- `payoff` payoff function.

The function returns a vector of **two** `DVector`s. The first element in the returned object is the equidistant spacial grid $(0 = S_0, S_1, \ldots, S_n = S_{\max})$. The second element is the approximate solution $(V_0^0, \ldots, V_n^0)$ at time $t_0 = 0$, where $V_i^0 \approx V(0, S_i)$.

The algorithm first computes $V_i^m = f(S_i)$ for $i = 0, \ldots, n$, where f denotes the payoff function. Then, using the tridiagonal solver, the algorithm recursively solves the system

$$AV^{j-1} = V^j$$

for $j = m, \ldots, 1$, where A is the tridiagonal matrix defined by (5).

Question 4

This question is based on `MClib`; `DOPlib` is also required.

The header file `SampleExam2015Q4Exercises.hpp` contains the declaration of the class

`exam_payoff`

which is a particular derived class from

`mc::payoff<mc::bvector>`

Given a trajectory of a solution path

$$Y_{t_1}, \ldots, Y_{t_{2^M}}, \text{ with } t_{2^M} = T, \text{ and } Y_t = [Y_t^1, \ldots, Y_t^N]$$

the payoff returns a one dimensional vector $X = [X^0]$, such that

$$X^0 = \sum_{k=1}^{2^L} \max\left[0, \left(\max_{1\le j\le N;\ \ Y_{k\Delta}^j > F} Y_{k\Delta}^j\right) - Ke^{Rk\Delta}\right]$$

where $\Delta = T/2^L$.

Implement the following member functions of the class.

```
exam_payoff(mc::scalar strike, mc::scalar rate, mc::scalar fl, unsigned int
    iSamplingAccuracy);
```

where the arguments are specified as follows.

- `strike` is the strike K,
- `rate` is the rate R,
- `fl` is the floor F and
- `iSamplingAccuracy` sampling accuracy L (we assume $L \le M$).

```
mc::bvector & operator()(path_out & poArg,
            mc::bvector & bvOut);
```

```
unsigned int SizePayoff() const;
```