C++ For Financial Engineering; Exercises

Steve Miller and Brad Baxter

July 2012

1 How To...

1. Add a new method to `TestClass`: open the file `TestClass.h` and add the header for the method:

   ```
   void TestNM();
   ```

   the add the body of the method to the file `TestClass.cpp`:

   ```
   void TestClass::TestNM()
   {
     // code in here...
   }
   ```

   Don't forget to put the code into `main()` to call the new method.

2. Compile the project: Use the "file" menu and select "build solution".

3. Run the project: Use the "Debug" menu and select "start without debugging".

4. Add a file to the project. Right-click on the project name in the left hand window and select "Add New Item" and then select the type of item, e.g. header file, cpp file and so on.

5. Add a new class to the project. Right-click on the project name and select "Add new item" and select "Class". Then select "C++ Class" and give a name. The `.h` and `.cpp` files will automatically be created and added to the project.

2 Basic Concepts

2.1 A pricing function

Background: Intrinsic data types, the math library and simple functions.

Task: You should now be able to a write a function to return the price of Euro call using the Black-Scholes formula:

$$C(S) = S\Phi(d_1) - K\mathrm{e}^{-rT}\Phi(d_2) \tag{1}$$

where S is the stock price today, K is the strike price, T is time to expiry, r is the risk free rate and

$$\begin{aligned} d_1 &= \left\{\ln(S/K) + (r + \sigma^2/2)T\right\}/\sigma\sqrt{T}, & (2)\\ d_2 &= \left\{\ln(S/K) + (r - \sigma^2/2)T\right\}/\sigma\sqrt{T} & (3)\\ &= d_1 - \sigma\sqrt{T} & (4) \end{aligned}$$

where σ^2 is the volatility. The function $\Phi(x)$ is the unit normal c.d.f. and is supplied.

The parameters should be the arguments to the function, and the return value should be the price. Suppose you were required to create a function which computes the price of the call and the put options at the same time (using put-call parity, $C(S) - P(S) = S - K\mathrm{e}^{-rT}$); what would you do?

2.2 A second pricing function

Background: Functions and arrays

Task: Write a function that computes the price of a Euro put or call using the B-S formula as above, but this time for every one of a set of strike price values passed into the function as an array of given length. Suppose that the calling program specifies the strike prices as something like:

```
unsigned int m = 10;
double *pA = new double [m];
pA[0] = 1.0;
pA[1] = 2.0;
// etc...
```

Naturally you can make use of the solution to the previous problem to do this.

2.3 Some linear algebra

Background: Arrays

Task: Write a function to compute the inner product ($c = \sum_i a_i b_i$) of two real-valued vectors (of the same length) that are passed as arguments. The arrays can be assumed to be specified in the calling program in the following way:

```
unsigned int m = 100; // for example
double *pA = new double [m];
double *pB = new double [m];
// set some values for the arrays
// and then call the function
```

3 Use of classes

3.1 Another Black-Scholes solver

Background: Basic class concepts

Task: Create a class to represent the Euro option. It should have data members to store all parameters required to solve the Black-Scholes formula, and methods to set and return these quantities. In addition, there should be methods to compute the put and call prices (and then store these data internally); you should be able to return these values when requested.

3.2 Interface and Implementation

Background: Basic class concepts, construction and copying

Task: Complete the implementation of the class `Test02` discussed in section 3.7

3.3 Example using inheritance

Background: Inheritance, virtual and pure virtual functions. Abstract classes.

Task 1: Create an abstract class for the payoff function for an option. It should have a pure virtual method called `Evaluate` which takes a double

value as an argument and evaluates the payoff function at this argument. It should also have a virtual destructor.

Task 2: Create concrete classes for put and call payoffs. Parameters of these payoff functions (e.g. strike price) can be passed in as arguments to the derived class constructor.

4 Use of the STL

4.1 Some linear algebra

Background: Basic STL concepts, use of the vector container

Task: Write a function that takes two double valued vector arguments and returns their dot product.

4.2 Some statistics

Background: Basic STL concepts, the `pair<>` container

Task 1: Write a function to compute the mean and variance

$$\hat{\mu} = \frac{1}{n}\sum_i a_i, \quad \hat{\sigma}^2 = \frac{1}{n-1}\sum_i (a_i - \mu)^2 \tag{5}$$

of a `vector<double>` argument, to return both of these values as elements of a `pair<double, double>` object. We wish to use it in the following way:

```
vector<double> oVec(m, 0.0);
// do something to fill up values of oVec
pair<double, double> oResults = SomeFunction(oVec);
cout << "Mean = " << oResults.first
  << "; var = " << oResults.second << endl;
```

Task 2: Now create a function to compute the covariance of two `vector<double>` arguments of the same length, using

$$\mathrm{Cov}(a,b) = \frac{1}{n-1}\sum_i (a_i - \mu_a)(b_i - \mu_b) \tag{6}$$

where μ_a and mu_b are the means of a and b respectively.

5 Monte Carlo

In the notes, it is shown how to generate normally distributed random numbers. The code for this course includes a basic random number generator. It can be called in the following fashion:

```
RNG oRNG;
double dVal = oRNG.Uniform01(); // draw from U(0,1)
double dVal2 = oRNG.UnitNormal(); // draw from N(0, 1)
```

WARNING: This rng uses system `rand()` which is generally not a good idea. It is not thread safe and on some operating systems might not be a good random number generator (in some statistical sense). This is provided for demo purposes only.

5.1 Basic Monte Carlo

Background: Use of classes

Task: Write a class to price Euro call and put using Monte Carlo method. The approach is to approximate the expectation of the payoff function using

$$C(S) \approx \mathrm{e}^{-rT} \frac{1}{n} \sum_{i=1}^{n} \max\left(S_T^{(i)} - K, 0\right) \tag{7}$$

where

$$S_T^{(i)} = S \exp\left\{\left(r - \sigma^2/2\right) T + \sigma\sqrt{T}\, Z^{(i)}\right\} \tag{8}$$

where $Z^{(i)} \sim N(0,1)$, $i = 1, ..., n$ i.i.d.

5.2 More Monte Carlo

Background: Use of classes, inheritance example above, monte carlo pricing example above

Task: An option can in principle have a completely arbitrary payoff function. Modify your option pricing class to accept a pointer to a payoff function object as defined above, and use its `Evaluate()` method in place of the explicit put or call payoff function.

5.3 Use of TNT

Read the section of the notes on TNT.

1. Check that when a LU decomposition has been performed, that the results are numerically correct. That is, if we recombine the upper and lower triangular matrices which are computed by TNT, is the original matrix recovered? Are there errors? At what level of precision do they appear?

2. Use an LU decomposition and the subsequent solution of the equations for a specific right hand side vector to compute the inverse of a matrix.

3. Use the cholesky factorisation of a real-symmetric positive definite matrix (i.e. a covariance matrix) together with some mean vector to generate draws from a multivariate Gaussian distribution.