Financial Programming in C++: Homework Assignment 3

Fall 2025,

MSQF, Fordham University

Due: September 18th, 2025

Problem 0

Unzip the file `HW3.zip` in a folder of your choice.

The folder should contain the following files:

- `discount.h` and `discount0.cpp`
- `forward.h` and `forward.cpp`
- `black_scholes.h` and `black_scholes0.cpp`

Problem 1: Swap Present Value

The present value of a **receiver swap** is given by the following formula:

$$\mathrm{pv} = \sum_{i=1}^{N-1} R\,\mathrm{d}T\,\mathrm{df}(t_i) + R\,\mathrm{d}t\,\mathrm{df}(T) + \mathrm{df}(T) - 1$$

The present value of a **payer swap** has the opposite sign:

$$\mathrm{pv} = -\sum_{i=1}^{N-1} R\,\mathrm{d}T\,\mathrm{df}(t_i) + R\,\mathrm{d}t\,\mathrm{df}(T) - \mathrm{df}(T) + 1$$

where all the variables have the same meaning we have seen in class:

- R is the swap rate,
- T is the maturity of the swap
- t_i are the payment dates of the swap
- $\mathrm{df}(t)$ is the discount factor at time t.
- $\mathrm{d}T = \frac{1}{\mathrm{freq}}$ is time between regular swap payments
- $\mathrm{d}t$ is the time between the last swap payment and the swap maturity

Problem 1.1

Make a copy of the file `discount0.cpp` and name it `discount.cpp`.

In this file **implement** the function `swap_pv` as declared in the file `discount.h`:

```
double swap_pv(bool is_receiver,double R, double T, double freq,  double r);
```

Where the function arguments are:

- a boolean flag `is_receiver` that is `true` if the swap is a receiver swap and `false` if the swap is a payer swap.
- the swap rate R
- the swap maturity T in years.
- the payment frequency freq
- the discount rate r

The function must return the present value of a swap as defined by those parameters.

Problem 1.2

Create a file named `test_discount.cpp`.

For every maturity starting with $T = 0.25$ years (one quarter) and up to $T = 30$ years (inclusive) in quarterly increments:

1. compute the swap rate R for that maturity.
2. verify using the function `swap_pv` that a **receiver swap** with that maturity and swap rate has present value equal to zero *within the numerical precision* of a `double` variable.

Assume that the swap frequency is semi-annual (`freq` = 2) and the discount rate r is 0.06.

Your program should output:

1. A header line with the following fields: `T, R, pv`.
2. One line for each maturity with the values of T, R and `pv` separated by commas.

Problem 2: Option Greeks and Multiple Function Returns

Option values are sensitive to changes in the market environment: the underlying price, the volatility, the level of interest rates, etc.

The **option greeks** are a set of measures that quantify the **risk exposure** (sensitivity) of the option value to changes in the market environment.

The most commonly used option greeks are defined as follows:

Delta the derivative of the option value with respect to the underlying price.

We will compute it using a **finite difference** approximation:

$$\Delta = \frac{V(S + \delta S, r, d, \sigma) - V(S, r, d, \sigma)}{\delta S}$$

where $V(S, r, d, \sigma)$ is the value of the option (as computed by `bs_price`), δS is a small change in the underlying price, by convention $\delta S = 0.01S$ (1% change in spot price).

Gamma is the derivative of the delta with respect to the underlying price. Using again a finite difference approximation we can compute it as:

$$\Gamma = \frac{V(S + \delta S, r, d, \sigma) + V(S - \delta S, r, d, \sigma) - 2V(S, r, d, \sigma)}{\delta S^2}$$

Vega is the change of the option price when volatility increases by 1%:

$$\text{Vega} = V(S, r, d, \sigma + \delta\sigma) - V(S, r, d, \sigma)$$

where $\delta\sigma = 0.01$.

DV01 is the change of the option price when the interest rate increases by 1 basis point (0.01%):

$$\text{DV01} = V(S, r + \delta r, d, \sigma) - V(S, r, d, \sigma)$$

where $\delta r = 0.0001$.

Problem 2.1

Copy the file `black_shcholes0.cpp` to a file named `black_scholes.cpp`.

In the file `black_scholes.cpp` implement the function `bs_risk` with signature:

```
double bs_risk(bool isCall, double K, double T,
               double S, double r,
               double d, double sigma,
               double &delta, double &gamma, double &vega, double &rho);
```

(as declared in file `black_scholes.h`).

The function must set the option price as the option's return value and set the greek values Δ, Γ, Vega and DV01 in the arguments provided **by reference**.

Problem 2.2

Create a file named `test_bs_risk.cpp`.

In the file `test_bs_risk.cpp` write a program that computes the option price and the option greeks for a Call option with one year maturity and for strikes ranging between $K = 50$ and $K = 200$ (inclusive) in increments of $K = 10$.

Assume that the market conditions are:

- Spot price $S = 110$
- Interest rate $r = 0.054$
- Dividend yield $d = 0.03$
- Volatility $\sigma = 0.25$

The program output should be a comma separated file a header line and a separate row for each strike K with the following fields:

- Strike price K
- Option value
- Delta Δ
- Gamma Γ
- Vega
- DV01

Problem 3: Fibonacci Numbers and Recursive Functions

The Fibonacci numbers are defined **recursively** as follows:

$$F_n = F_{n-1} + F_{n-2}$$

where $F_0 = 0$ and $F_1 = 1$.

0.1 3.1

Create a file `fibonacci_recursive.cpp`.

In the new file Write a recursive function:

```
long long fibonacci(long long n) {
```

using the definition above.

Fibonacci numbers grow very fast, so we use a `long long` to make sure we have enough range.

Problem 3.2

In the same file `fibonacci_recursive.cpp` write a `main` function that:

1. Prompts the user for an integer n.
2. Reads the user input from the terminal using the standard input stream `std:cin`.
3. Outputs the value of F_n to the terminal.

Test the program with small values of $n \approx 10$.

Problem 3.3

The Fibonacci numbers can be computed with a loop using the following algorithm:

1. Initialize variables $a = 0$ and $b = 1$ to the first to Fibonacci numbers.
2. For $i = 2, 3, \ldots, n$ set

$$\begin{aligned} c &= a + b \\ a &= b \\ b &= c \end{aligned}$$

   so that $a = F_{i-1}$ and $b = F_i$ at each step.
3. The value of F_n is stored in b at the end of the loop.

Create a file `fibonacci.cpp` and implement the algorithm above in a function:

```
long long fibonacci(long long n) {
```

Re-implement the `main` function as in Problem 3.2 but calling the new implementation of the `fibonacci` function.

Test the program with small values of $n \approx 10$ and verify that they return the same values.

Problem 3.4

Try now to compute F_{45} using the recursive function and the loop function.

Compare their performance. Which one is faster? Can you explain why?

Try computing F_{100} with both functions. What happens?

Submissions

You must submit a zip file to complete this homework. The zip file must contain the following files:

1. Problem 1:
   (a) `discount.cpp` with the implementation of the `swap_pv` function.
   (b) `test_discount.cpp` with the implementation of the test program.
   (c) `test_discount.txt` with the output of running the executable compiled from `test_discount.cpp`.
2. Problem 2:

(a) `black_scholes.cpp` with the implementation of the `bs_risk` function.

(b) `test_bs_risk.cpp` with the implementation of the test program.

(c) `test_bs_risk.txt` with the output of running the executable compiled from `test_bs_risk.cpp`.

3. Problem 3:

   (a) `fibonacci_recursive.cpp` with the implementation of the recursive function.

   (b) `fibonacci.cpp` with the implementation of the iterative function.

   (c) The output of running the executable compiled from `fibonacci.cpp` with $n = 45$.

   (d) Write (in a comment) you answer to **Problem 3.4** at the end of the file `fibonacci.cpp`.