Financial Programming in C++: Homework Assignment 5

Fall 2025,
MSQF, Fordham University

Due: October 2nd, 2025

Problem 0

Upzip the `HW5.zip` file.

It should contain the following files:

- `solve.cpp` and `solve.h`: the bisection solver discussed in class.
- `discount0.cpp` and `discount.h`: the input file for fixed income functions and its header file.
- `test_bond_bootstrap0.cpp`: the test program for the bond bootstrap.
- `forward.h` and `forward.cpp`: for equity forward calculations functions and its header file.
- `volatility.h` and `volatility.cpp`: for volatility curve functions and its header file.
- `black_scholes.h` and `black_scholes0.cpp`: for Black-Scholes functions and its header file.
- `test_vol_bootstrap0.cpp`: the test program for the volatility bootstrap.

Problem 1: Bond Prices

The present value (fair market price) of a bond is given by the formula:

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

where the sum runs over **regular payments**, $\mathrm{d}f(t)$ is the discount factor at time t, $\mathrm{d}T = \frac{1}{\mathrm{freq}}$ is the time between **regular** payments and $\mathrm{d}t$ is the fraction of a year of the last stub payment period.

This function is implemented in the file `discount.cpp`. The signature of the function (declared in file `discount.h`) is

```
double bond_pv(double T,double coupon, double freq,
               int tenors, const double *Ts, const double *rs);
```

Problem 1.1: Bond Price Function

Rename the file `discount0.cpp` to `discount.cpp` and `test_bond_bootstrap0.cpp` to `test_bond_bootstrap.cpp`.

In Problem 1.2 you will implement bootstrap function with signature:

```
int bond_bootstrap(int tenors, double *Ts, double *coupons,
                   double *prices,double freq, double * rs,
                   int max_iter, double tol) {
```

that will solve for the zero coupon rates `rs` of the discount curve given the bond prices `prices`.

As a helper, in Problem 1.1 you will **implement the bond price wrapper function** with signature:

```
double bond_function(double r, int size, double *params)
```

where the parameters are mapped as follow:

- `params[0]` is the the **index** of the bond expiry we are solving for in the bond expiry array `Ts` of `bond_bootstrap`.
- `params[1]` is the coupon of the bond.
- `params[2]` is price of the bond.
- `params[3]` is the coupon payment frequency.
- `params[4+k]` is the expiry of the k-th bond used to bootstrap the discount curve, for the $k = 0, 1, \ldots, \text{tenors} - 1$ bonds.
- `params[4+N+k]` is the zero coupon bond rate of the k-th expiry (this values will get determined as the bootstrap progresses).

HINT: See the functions `swap_function` and `swap_bootstrap` in the file `discount.cpp` for an example of how to use the `params` array.

Problem 1.2: Bond curve bootstrap

In the file `discount.cpp` implement the bootstrap function:

```
int bond_bootstrap(int tenors, double *Ts, double *coupons,
                   double *prices,double freq, double * rs,
                   int max_iter, double tol) {
```

That iterates over a number `tenors` of bonds and bootstraps the zero coupon rates into the array `rs` so that

$$\text{price}_i = \texttt{bond\_price}(T_i, c_i, \texttt{freq}, \texttt{rs})$$

for $i = 1, \ldots, \texttt{tenors}$, expiries T_i and coupon c_i.

The caller of the function is responsible to allocate space for the array `rs`.

Problem 1.3: Test Bond Bootstrap

Rename file `test_bond_bootstrap0.cpp` to `test_bond_bootstrap.cpp`.

In the file `test_bond_bootstrap.cpp`:

- bootstrap the bonds details provided in the `main` function.
- For every input expiry T_i output the zero coupon rate and the bond present value discounted with the bootstrapped curve.
- Verify that the present value of the input bonds match the input prices. If not, output a clear error message and exit program with a non-zero error code.

Problem 1.4: Compile and Run

Compile and run the program `test_bond_bootstrap.cpp`.

Capture the output of the program in a file `test_bond_bootstrap.csv`.

Problem 2: Volatility Curve Bootstrapping

Given a set of (discounted) option prices, we would like to bootstrap a volatility curve able to reprice the options.

Problem 2.1

Implement the function:

```
int volatility_bootstrap(int tenors,
                         const bool * is_call, const double *Ks, const double *Ts,
                         double S, const double * rs, const double *ds,
                         const double *prices,
                         double *sigmas,
                         int max_iter=100, double tol=1e-8);
```

See file `black_scholes.h` for documentation on the interpretation of inputs.

HINT: The calculation of option implied volatilities is implemented by function `bs_implied_vol` in the same header file.

Problem 2.2

Volatilities must satisfy a **no arbitrage condition**:

$$\sigma^2(T')T' \geq \sigma(T)^2T$$

whenever $T' > T$

In function `volatility_bootstrap` implement a check for this condition for all input option expiries T_i. Return a negative status value from `volatility_bootstrap` if the volatility curve is arbitrageable.

Problem 2.3

We will now test the volatility bootstrap function.

Rename file `test_vol_bootstrap0.cpp` to `test_vol_bootstrap.cpp`.

Using the **swap rates** provided in the `main`'s array `Rs` of file `test_vol_bootstrap.cpp`, bootstrap the zero coupon rate curve.

Problem 2.4

In the file `test_vol_bootstrap.cpp`, the option prices for multiple volatility curves are provided simultaneously:

```
prices[i]
```

is an array with `tenors` option prices for the *i*-th volatility curve.

Using the operator `sizeof` compute the number of volatility curves provided in the `main`'s array `prices`.

For each one of the option price arrays:

- bootstrap the volatility curve using the function `volatility_bootstrap`, the zero coupon rate curve computed in Problem 1.2 and the other market data provided in `main`.
- If the bootstrap process failed, output a clear error message, letting the user know what curve failed to bootstrap and continue processing the next curve.
- If the bootstrapped curve is arbitrageable, output a clear error message, letting the user know what curve contains an arbitrage and continue processing the next curve.
- If the bootstrapped curve is not arbitrageable, output the bootstrapped volatilities in a comma separated line.

Problem 2.5

Compile and run the program `test_vol_bootstrap.cpp`. Capture the output of the program in a file `test_vol_bootstrap.csv`.

Submission

You must summit a zip file containing the following files to complete this assignment:

1. `discount.cpp`, `test_bond_bootstrap.cpp` and `test_bond_bootstrap.csv` with your answer to Problem 1.
2. `black_scholes.cpp`, `test_vol_bootstrap.cpp` and `test_vol_bootstrap.csv` with your answer to Problem 2.