How to run the program

1. Download the repository "ImpliedVolatilityEngine"and then open the Visual Studio file "OptionImpliedVol.sln".
2. Go to the "main.cpp" file and run the program, it should take around 1 minute to run.
3. Output data will be visible in "ImpliedVolatilityEngine/ImpliedVolatilityEngine/outputData"

Note: .zip extraction takes some time as I added the necessary boost library to the repository (https://github.com/pa12g10/ImpliedVolatilityEngine) to ensure the program runs.

Application Dependencies

The program makes use of the latest boost package (Version 1.69.0) see link below:
https://www.boost.org/users/download/

The library has been attached to the repository to ensure the program has the necessary dependencies to run.

The main #includes used are:

```
#include <boost/lexical_cast.hpp>
#include <boost/bind.hpp>
#include <boost/function.hpp>
#include <boost/test/included/unit_test.hpp>
#include <boost/test/tools/assertion.hpp>
#include <boost/math/constants/constants.hpp>
#include <boost/math/distributions/normal.hpp>
```

Source Files

- Main.cpp – This is the main program to run the task stated in the test. Taking the input trade data and returning the implied volatility.
- testSuite.cpp – This is where the test cases are ran, to run them you must comment out the main.cpp and then uncomment the testSuite.cpp. Test cases are described below.
- BrentSolver.cpp/BrentSolver.hpp – These files are the main numerical section of the program, a modified Brent Solver which finds the implied volatility. It takes a generic functor in this case the option price function from ImpliedVolatility.cpp and attempts to find the volatility such that the option price matches the functors solution.
- ImpliedVolatility.cpp/ImpliedVolatility.hpp – These files encapsulates all the attributes and functions for option pricing relative to this test. Catering for two underlying types {Future, Stock} and models {Black, Bachelier} and finally Put or Call types.
- ImpliedVolatilityExtractor.cpp/ImpliedVolatilityExtractor.hpp – These files take care of the data extraction and triggering of the Brent Solver to find the implied volatility and finally triggering the output to be written to a .csv file.

Unit Tests

The following items are checked to give comfort towards the validity of the results:

- `test_statistics` – This test case looks to see if the **Standard Normal CDF** is returning ½ when an input of zero is used.
- `test_openFile` - This test checks **opening the input file** and the row count is equal to the total number of trades provided from input.csv.
- `test_BrentSolver` **- Brent solver** I take a quadratic (x+1)*(2x-1) with roots (1/2, -1) clearly visible from the analytical solution and return one of the roots given an interval [`0.0001,2`]
- `test_OptionPricing` **-Implied volatility** to verify I am returning the correct volatilities I price an option with volatility equal to 20% and using this option price I use the Brent solver to return the same volatility used at the beginning.

Observations/Remarks

- Max Iteration from the Brent solver never seems to go above 40 so I have capped it to improve computing time.
- The implied volatility occurs as 'nan' sometimes as the Brent solver can no longer find a large enough volatility to match the option price given. This is because the normal CDF is converging and little changes in volatility is no longer causing changes in the option price. Below graph is from trade ID 39 you can see as the option price increases so does the volatility to match until it reaches a peak beyond this point the CDF will return the same value.