Source Code README

This document is about the C++ implementation of the paper "A fast implicit method for time-dependent Hamilton-Jacobi PDEs" by Vladimirsky and Zheng. Both the paper and the source code can be downloaded from the project webpage.

1 Code Compilation

The code compilation has been tested in Linux using both gcc and Intel's compiler. If you have CMake installed, you can compile it using the following steps:

1. Create a build dir and go into it

   ```
   mkdir gcc-build && cd gcc-build
   ```

2. Run cmake to configure the Makefile

   ```
   cmake ..
   ```

3. Compile it

   ```
   make
   ```

After compiling the code, you can run the three executables located in gcc-build/src folder. They are ExplicitTDHJ2D, ImplicitTDHJ2D and MixedTDHJ2D respectively corresponding to the explicit, implicit and hybrid methods described in the paper. Different command line options are possible. See their details by providing the -h option (e.g., run ImplicitTDHJ2D -h).

2 Required Library

This code requires Boost library installed.

3 Customize the Tested PDEs

The code solves the time-dependent HJB PDEs which have the form

$$u_t + f(\boldsymbol{x}, t)|\nabla u| = g(\boldsymbol{x}, t).$$

We implemented three different numerical solves, namely the explicit, implicit and hybrid methods. Their main code are respectively ExplicitTDHJ2D.cpp, ImplicitTDHJ2D.cpp and MixedTDHJ2D.cpp. All these solvers are implemented as C++ template classes. The f and g functions and the boundary conditions are specified as the template parameters.

Both the f and g functions are defined as a C++ functional. Namely, a C++ struct with operator (). For example, a f function is defined as

```
struct func_F
{
    inline double operator() (const vector2d& pos, double t) const
    {   return ... }
};
```

The boundary condition class should specify the number of boundary nodes, the positions of the boundary nodes, and their values. Please look at the code in src/BoundaryCond.h as examples.

Once you have your own f, g and boundary condition classes are defined. You can use them by defining a test case in the solver's main code (e.g. ImplicitTDHJ2D.cpp),

```
#ifdef USE_TEST_XX
typedef SimpleBoundaryCond2D                        TBC;
typedef CachedImpLattice2D<func_F, func_G, TBC>     TLattice;
```

Here USE_TEST_XX is the test case number. You can now enable this test case by define a flag USE_TEST_XX in src/config.h.