User's guide for the ROC-HJ * solver: Finite Differences and Semi-Lagrangian methods

February 2013
Version 1.0

Current developers : O. Bokanowski, A. Desilles, H. Zidani.

1 Problem solved

This is a C++ MPI/OpenMP library for solving d-dimensional Hamilton-Jacobi-Bellman equations by finite difference methods, or semi-lagrangian methods. The equations considered are of the following type.

1.1 Evolutionary case

The problem is to find $u = u(t,x)$ solution of

$$\begin{cases} \frac{\partial u}{\partial t} + H(x, \nabla u) = 0, \quad x \in \Omega, \quad t \in [0,T], \\ u(t,x) = u_{bord}, \quad x \in \partial\Omega, \quad t \in [0,T], \\ u(0,x) = u_0(x), \quad x \in \Omega \end{cases} \tag{1}$$

where Ω is a domain of $\mathbb{R}^d$ of the form $\prod_{i=1}^d [a_i, b_i]$, and u_{bord} is a constant (other boundary conditions are possible). The function $H(x,p)$ can be defined either directly or as follows [1]

$$H(x, \nabla u) := \max_{a \in \mathcal{A}} \Big(- f(x,a) \cdot \nabla u - \ell(x,a) \Big), \tag{2}$$

where A is a set of controls, of the form $\prod_{i=1}^{n_A} [\alpha_i, \beta_i]$, or [2]

$$H(x, \nabla u) := \max_{a \in \mathcal{A}} \min_{b \in \mathcal{B}} \Big(- f(x,a,b) \cdot \nabla u - \ell(x,a,b) \Big), \tag{3}$$

*Reachability, Optimal Control, and Hamilton-Jacobi

1. $\max_{a\in\mathcal{A}}(\cdots)$ can be also $\min_{a\in\mathcal{A}}(\cdots)$
2. $\max_{a\in\mathcal{A}} \min_{b\in\mathcal{B}}(\cdots)$ can be also $\min_{a\in\mathcal{A}} \max_{b\in\mathcal{B}}(\cdots)$

where $\mathcal{A}$ and $\mathcal{B}$ are control sets of the form $\prod_{i=1}^{n_A}[\alpha_i^A, \beta_i^A]$ and $\prod_{j=1}^{n_B}[\alpha_j^B, \beta_j^B]$.

1.2 Stationary case

The problem is to find $u = u(x)$ solution of

$$u + H(x, \nabla u) = 0, \quad x \in \Omega \backslash \mathcal{C}, \tag{4}$$

with H given by (2), Ω a square of $\mathbb{R}^d$ of the form $\prod_{i=1}^{d}[a_i, b_i]$, and $\mathcal{C}$ is a given subset of Ω. In that case, the boundary conditions are given by

$$u(x) = 0, \quad x \in \partial\mathcal{C}.$$

and

$$u(x) = u_{bord}, \quad x \in \partial\Omega,$$

and the set $\mathcal{C}$ is also coded by the function u_0, such that $\mathcal{C} := \{x,\ u_0(x) \leq 0\}$.

2 Compilation and execution (under GCC and CMake)

In order to make the program works, the c++ compiler GCC and the build system CMake are proposed to help the compilation process. The source files are typically in the directory `src/`. User's data files are typically in the directory `data/`.

For deeper informations, we redirect the user to the following links :
http ://gcc.gnu.org/install/
http ://www.cmake.org/cmake/resources/software.html

Building the Makefile : `cmake .` (do not forget the "dot")

Compilation : `make` (source `.cpp` files are generally in the directory `src/`)

Options : `make clean` (cleaning some `.o` files), `make cleanall`.

Other : `"./clean"` to first erase all unecessary files (can be done before the `"cmake ."` command.)

Sequential code Execution (basic) :

```
./exe
```

Execution (with options)

```
./exe -nn NN -nc NC
```

Options :

- option -nn `NN` : number of mesh points per dimension
- option -nc `NC` : number of controls per command's dimension.

OpenMP Some options run with OpenMP. Execution :

```
./exe -nt nbth
```

where `nbth` is the number of threads (typically try `nbth=2` or `nbth=4`).

3 User's parameters

The file `data/data_simulation.h` is the main user's input file and contains the parameters that define the equation to be solved. It can include an other `data_xxx.h` file (see basic examples `data_basicmodel.h`, `data_FD_2d_ex1_basic.h`). It contains also parameters relative to the output during the execution saving data and data post-treatment (see paragraph "Output parameters").

- `NAME` : name of the problem
- `DIM` : dimension d of the problem

Choice of the solver

- `METHOD` $\in$ `{MDF,MSL}` :
  `MDF` : Finite Difference Method
  `MSL` : Semi-Lagrangian Method

Stopping criteria

- `T` : terminal time
- `MAX_ITERATION` : to stop the program when this maximum number of iterations is reached.
- `EPSILON` : for stopping criteria. If set to `0`, do nothing. If set to some positive value ϵ, then the program will stop at iteration $n+1$ as soon as

$$(\Delta x_1 \cdots \Delta x_d) \sum_i |v_i^{n+1} - v_i^n| \leq \epsilon.$$

  (the L^1 error between two successive steps is smaller than ϵ).

Finite Difference Method This method is used when `METHOD=MDF`, and it utilizes :

- `COMMANDS` $\in$ `{0,1,2}` :
  This decides how we define the Hamiltonian function $H(x,p)$ (using the "commands" or not).
  `1` : the commands are used in the definition of the numerical hamiltonian, as in (2). The program will use a numerical hamiltonian function corresponding to a finite difference approximation of $H(x, \nabla u) = \max_\alpha(-f(x,\alpha).\nabla u - \ell(x,\alpha))$. Examples are given in `data_basicmodel.h` (rotation), or `data_FD_2d_ex1_basic.h` (eikonal equation).

`2` : the commands are used in the definition of the numerical hamiltonian, as in (3). The program will use a numerical hamiltonian function corresponding to a finite difference approximation of (for instance) $H(x, \nabla u) = \max_{\alpha} \min_{\beta}(-f(x, \alpha, \beta).\nabla u - \ell(x, \alpha, \beta))$.

`0` : an `Hnum` function must be directly defined by the user ; it has to be a numerical hamiltonian corresponding to $H(x, p)$. Examples are given in `data_advancedmodel.h`, `data_FD_2d_ex1_advanced.h`.

If `COMMANDS=1`, we need also to define

- function `dynamics` : $f(x, \alpha)$
- function `distributed_cost` : $\ell(x, \alpha)$
- a set of commands corresponding to the discretization of a cube $\prod_{i=1,n_c}[\alpha_i, \beta_i]$, defined by `cDIM` (dimension n_c of the set of controls), `NCD[cDIM]` (number of commands per direction) `UMIN[cDIM], UMAX[cDIM]` (min and max values α_i and β_i of the commands in each direction).

If `COMMANDS=2`, we need to define

- function `dynamics2` : $f(x, a, b, t)$
- function `distributed_cost2` : $\ell(x, a, b, t)$
- two sets of commands $\mathcal{A}$ and $\mathcal{B}$, corresponding to the discretization of $\mathcal{A} \equiv \prod_{i=1,n_A}[\alpha_i^A, \beta_i^A]$ and $\mathcal{B} \equiv \prod_{i=1,n_B}[\alpha_i^B, \beta_i^B]$. The parameters `cDIM, cDIM2` will correspond to n_A and n_B, the dimension of each set of controls), `NCD[cDIM]` is the number of commands per direction, and `UMIN[cDIM], UMAX[cDIM]` are the min and max values of α_i and β_i of the commands in each direction.

If `COMMANDS=0`, we need to define

- function `Hnum` : a first order numerical Hamiltonian.
- function `compute_Hconst`. This function must define d constants which are bounds for $\left|\frac{\partial H}{\partial p_i}(x, p, t)\right|$ $(i = 1, \ldots, d)$, for $x \in \Omega$ and for possible gradient values p and time t. This is then used to set the time step Δt in order to satisfy a CFL condition. These constants may also be used in the function `Hnum`.
  (The user may also define directly the constants c_i =`Hconst[i]` and initialize the previous function accordingly.)

Then, in all cases, we have also to set the scheme discretisation parameters :

- `TYPE_SCHEME` ∈ `{LF,ENO2}` : type of spacial discretization
  `LF` : Lax-Friedrich scheme (first order scheme)
  `ENO2` : ENO scheme of second order to approximate the derivatives ∇u
- `TYPE_RK` ∈ `{RK1,RK2,RK3}` : Time discretization by a Runge-Kutta method of order 1, 2 or 3.
  `RK1` : RK method of order 1

RK2 : RK method of order 2
RK3 : RK method of order 3

Semi-Lagrangian Method This method is used when METHOD=MSL. It assumes that :

$$H(x, \nabla u) \equiv \max_{\alpha}(-f(x,\alpha).\nabla u - \ell(x,\alpha)).$$

Then it needs to define :

- function dynamics : dynamics $f(x,\alpha)$
- function distributed_cost : cost function $\ell(x,\alpha)$
- TYPE_SCHEME ∈ {STA,EVO} :
  STA : stationnary case, for solving (4). In this case, the scheme is based on an iterative procedure. [3]
  EVO : dynamic case, for solving (1)
  ORDER : this value is set to 1 for solving first order equations of type (1) (see other section below for second order HJB equations).
- TYPE_STA_LOOP ∈ {NORMAL, SPECIAL} : Mesh loop order during mainloop for the stationary case.
  NORMAL : normal ordering loop
  SPECIAL : special ordering loop (makes 2^d loops at each iteation, modifiying the current values of the data v during each loop)
- TYPE_RK ∈ {RK1_EULER,RK2_HEUN,RK2_PM} :
  RK1_EULER : RK1 Euler scheme
  RK2_HEUN : RK2 Heun scheme
  RK2_PM : RK2 Mid-point scheme
- INTERPOLATION ∈ {BILINEAR,PRECOMPBL,DIRPERDIR} :
  BILINEAR : (default value) bilinear interpolation ($Q1$ interpolation)
  PRECOMPBL : precompute the interpolation coefficients (to use with tiny mesh sizes)
  DIRPERDIR : another interpolation method
- COMPLEMENTS :

3. The scheme is based on the following iteration on $n \geq 0$, for a fixed $h > 0$:

$$u^{n+1}(x) + \max_{a}\left(\frac{u^{n+1}(x) - u^n(x + hf(x,a))}{h} - \ell(x,a)\right) = 0,$$

untill $\|u^{n+1} - u^n\|$ reaches a given threshold. Therefore we obtain the following recursion

$$u^{n+1}(x) = \min_{a}\left(\frac{1}{1+h}[u^n](x + hf(x,a)) + \frac{h}{1+h}\ell(x,a)\right)$$

P_INTERMEDIATE : number of intermediate timesteps to go from t_n to $t_n + \Delta t$ for computing a caracteristic for a given RK method.
ORDER $\in$ {1,2} : decide treatement for first order HJ equation or for second order HJ equation.

Discretization parameters

- ND[DIM] : tab containing the size mesh in each direction (cartesian mesh)
- MESH $\in$ {0,1} : default is 1. It utilizes ND[i]+1 points in direction i. (If MESH=0 the mesh points are at the center of the mesh cells, and there are ND[i] points in direction i)
- DT : the time step used for the solver, for the evolutive equation.
  - For the finite difference approach, if DT=0, then time step DT is computed so that the CFL condition be satisfied, that is, such that
  ```
  DT * (Hconst[0]/dx[0] + Hconst[1]/dx[1] + ... ) <= CFL
  ```
  where the CFL number belongs to $[0, 1]$. Otherwise, if DT>0, then the value DT is used for the time step.
  - For the semi-lagrangian approach and for the stationnary equation, the parameter h in the iterative procedure is also set to DT

Other parameters for the definition of the problem The parameters are :

- XMIN[DIM], XMAX[DIM] : the boundary (a_i, b_i) of the domain in each direction
- PERIODIC[DIM] : each componant set the periodicity for each direction
  - 1 : direction is periodic
  - 0 : direction is not periodic
- VBORD : constant eventually used in the case of Dirichlet boundary condition.
- function v0 : the initial data
- function Vex : if known, the user can put the exact solution of the problem here.
- integer EXTERNAL_v0 $\in$ {0,1} : if 0 initialize with function v0, otherwise initialize with last VF.dat values.

Obstacle terms Instead of solving $u_t + H(t, x, \nabla u) = 0$, the solvers can also treat HJ obstacle equations such as

$$\min\left(\frac{\partial u}{\partial t} + H(x, \nabla u),\ u - g(t, x)\right) = 0. \tag{5}$$

In particular it forces to have $u(t, x) \geq g(t, x)$ in the case of (5). For this, the following parameters are used :

- OBSTACLE $\in$ {0,1}
  - 0 : no obstacle taken into account.
  - 1 : Equation (5) is treated, with the obstacle g .

- function `g_obstacle`

Output parameters

- `COMPUTE_MAIN_LOOP` :
  `1` : the main computational loop (iterative scheme) is called
  `0` : the main loop is not called, only data initializations and savings are performed.
- `COMPUTE_VEX`∈`{0,1}` : will save a `VEX.dat` file.
  `1` : if the exact solution is given by the user (using function `Vex`)
  `0` : no saving.
  In particular it will compute errors (see below) relatively to this value.
- `COMPUTE_TMIN`∈`{0,1}` : will compute file `tmin.dat` which contains the first time $t_n \in [0, T]$ (or some $P1$ interpolant) such that $v(t_n, x) \leq 0$ (Set the value to `INF=1.e5` otherwise.)
- `PRECOMPUTE_COORD` :
  `1` : precomputes the coordinates : faster computations but more memory demanding.
  `0` : no precomputations.
- `SAVE_VFALL` ∈ `{0,1}` : to save the value of v each `SAVE_VFALL_STEP` iterations (into file `VFALL.dat`).
- `CHECK_ERROR` ∈ `{0,1}` : to compute error every `CHECK_ERROR_STEP` steps (L^∞ and L^1 error computations) Errors are relative to the `Vex` function. Error are computed only in the region of points x such that `|Vex(t,x)|<C_THRESHOLD` [4]
- `COUPE_DIMS[DIM]`, `COUPE_VALS[DIM]` :
  - list of integers n_i (0 or 1) to define the two variables used for the cut.
  - list of values (c_i or 0.) to define the position of the cut. The values c_i are used only when $n_i = 0$.
  - The output is in the file `output.dat`
  - Example : `COUPE_DIMS[DIM]={1,0,1}` and `COUPE_VALS[DIM]={0.,0.5,0.}` define a cut in the plane $x_2 = 0.5$.

4 Output files

• `VF.dat` contains the final u at the end of the computation. The file is structured as follows, on each line :

```
i1  i2  ....  id  val
```

where `val` corresponds to the value $u(T, x)$ at mesh point $x = (x_{i_1}, \ldots, x_{i_d})$.
• `VEX.dat` contains the final value as programmed in `Vex` (if `COMPUTE_VEX=1`).

4. For the moment, `C_THRESHOLD` is defined in `src/main.cpp`.

• `coupe.dat` (and `coupeex.dat`) can be obtained in the same way, for a problem in dimension $d \geq 3$ and when we make a "cut" in some particular subspace with some given coordinates, and that we want to visualize the results. See paragraph "Output parameters" for using `COUPE_DIMS` and `COUPE_VAL`. This is typically used to make a 2d cut (see for instance example `data/data_FD_3d.h`).

5 Graphic outputs

Matlab : the file `read.m` can be executed by typing `'read'` in the Matlab's user interface Some parameters are given below :

For the first figure :

- `level_set` : a level set value.
- `PLOT_CONTOUR`∈`{0,1}` : if set to `1`, will plot contours of the output with `level_set` value.
- `PLOT_REACHABLE`∈`{0,1}` : if set to `1`, will plot the 2d set of points where output value is $\leq$ `level_set`).

This plot is based on the output value which is in the file `VF.dat` for `DIM=2`, and a priori in the file `coupe.dat` for `DIM>2`.

For the second figure :

- `PLOT_3d`∈`{0,1}` : set this parameter to `1` in order to obtain a 3d graph of the result.
- `PLOT_TMIN`∈`{0,1}` : (default is `0`). Set this parameter to `1` in order to obtain a 3d graph of the minimal time (to reach a target). Will then use output `tmin.dat`, and will draw several contour plots ranging from 0 to T (plotting `tmin.dat` is only ok for `DIM=2`).

Also for both figures,

- `AXIS_EQUAL`∈`{0,1}`, if set to `1`, will make "axis equal",
- `TRAJECTORY`∈`{0,1}`, if set to `1`, will line-plot the list of first two components (x_1, x_2) of file "trajectory.dat".

Paraview : For the moment this option is only working for some 2d/3d cases.
Launch "paraview", then load the files in `VTK/*` (load `tab*` and so on).

6 Examples

Rotation example : We give two ways to program an advection-like example, in files `data_basicmodel.h` and `data_advancedmodel.h`. We consider the equation

$$\partial_t u + \max(0, -f(x) \cdot \nabla u) = 0.$$

with the parameters $\Omega = [-2,2]^2$, $T = 0.5$, $f(x_1,x_2) = 2\pi(-x_2,x_1)$. (Hence the Hamiltonian $H(x,p,t) = \max(0,-f(x)\cdot p)$.)

In the first way (`data_basicmodel.h`), we define dynamics

$$f(x,a) = af(x)$$

with two control values $a \in \{0,1\}$.

In the second way (`data_advancedmodel.h`), we define the Lax-Friedrich numerical hamiltonian H_{num} associated to H (see (8)).

Eikonal equation : see the files `data_FD_2d_ex1_basic.h` and `data_FD_2d_ex1_advanced.h` The problem solved is

$$\partial_t u + c(x,t)\|\nabla u\| = 0, \quad x \in [-2,2]^d, \quad t \in [0,T]. \tag{6}$$

$$u(0,x) = u_0(x), \quad x \in [-2,2]^d, \tag{7}$$

with $d = 2$, $T = 1$, here $c(x,t) \equiv 1$, and with some (radially symetric) initial data $u_0(x)$.

In the file `data_FD_2d_ex1_basic.h`, a scheme is programmed using a control-discretisation of $\|\nabla u\|$:

$$\|\nabla u\| \sim \max_{k=0,\ldots,\texttt{NCD}-1} \langle(\cos(\theta_k),\sin(\theta_k),\nabla u\rangle.$$

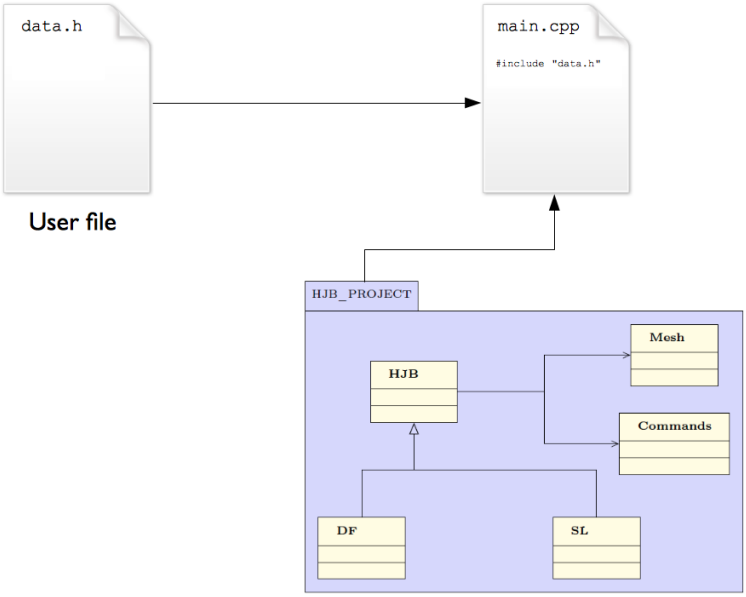
In the file `data_FD_2d_ex1_advanced.h`, a scheme is programmed using a Lax-Friedriech numerical approximation H_{num} associated to H :

$$H_{num}(x,(p_1^-,p_1^+),\ldots,(p_d^-,p_d^+),t) := H(x,\frac{p^-+p^+}{2},t) - \sum_{i=1}^{d} c_i\left(\frac{p_i^+ - p_i^-}{2}\right) \tag{8}$$

The exact solution is given for comparison.

7 Source architecture

In this section, we present the architecture used in the implementation of the project. The `HJB` class represents the main class and the `DF` and `SL` classes are sub-classes of `HJB`. `Commands` and `Mesh` are seperated classes. We remind that the user needs to code the problem parameters into a C++ header file and to include its name in the file `data/data_simulation.h`. For example, if the user data file is called `data_myexample.h`, then the line `#include "data_myexample.h"` (and only this one) must be included in the file `data/data_simulation.h`.


data.h
User file
main.cpp
#include "data.h"
HJB_PROJECT
Mesh
HJB
Commands
DF
SL