ROC-HJ: Reachability analysis and Optimal Control problems - Hamilton-Jacobi equations

Olivier Bokanowski*, Anna Désilles†, Hasnaa Zidani‡

February 2013

1 Introduction

The software ROC-HJ is a C++ library that implements a set of numerical methods for solving some Hamilton-Jacobi equations arising in optimal control theory. The library also contains some useful tools for analyzing the numerical solutions and aiming at designing the optimal control laws along with the corresponding optimal trajectories.

The code can run for any dimension $d \geq 1$ limited only by the machine capacity.

The library can be used for a large class of deterministic control problems including: reachability analysis, path planning, collision avoidance, infinite horizon control problems, minimum time problems, Mayer or Bolza type problem, state-constrained control problems, differential games, exit time problems. More precisely, it can be used to approximate the numerical solution of the following type of equations:

Finite horizon HJB equation: The problem consists of finding an approximation of the solution $u(t,x)$ of the PDE:

$$\begin{cases} \frac{\partial u}{\partial t} + H(t,x,\nabla u) = 0, \quad x \in \Omega, \quad t \in [0,T], \\ u(0,x) = u_0(x), \quad x \in \Omega \\ + \text{ Boundary conditions on } \partial\Omega \times (0,T), \end{cases} \tag{1.1}$$

where Ω is a domain of $\mathbb{R}^d$ of the form $\prod_{i=1}^d [a_i, b_i]$, $u_0(\cdot) : \mathbb{R}^d \to \mathbb{R}$ is a given function. So far, the code run with one of the following boundary conditions:

$$\text{Dirichlet type condition:} \quad u(t,x) = Cst = u_{\text{border}}, \tag{1.2a}$$

$$\text{a periodic condition:} \quad u(t, x+\Pi) = u(t,x), \tag{1.2b}$$

$$\text{or} \quad \partial_{xx} u(t,x) = 0. \tag{1.2c}$$

The boundary condition can be also a combination of different types. Actually, one may consider a periodic condition on a part of the space vector and a Dirichlet-type condition on the remaining

*Université Paris Diderot, Laboratoire Jacques Louis Lions. Email: boka@math.jussieu.fr
†ENSTA ParisTech, Unité de Mathématiques Appliquées. Email: Anna.Desilles@ensta-paristech.fr
‡ENSTA ParisTech, Unité de Mathématiques Appliquées. Email: Hasnaa.Zidani@ensta-paristech.fr

space components. For example, the solution of (1.1) may satisfy periodic conditions with respect to the l first components with a period of $(\pi_1, \cdots, \pi_l)$ and a Dirichlet-type condition on the $(d-l)$ remaining components:

$$u(t, x_1 + \pi_1, ..., x_l + \pi_l, x_{l+1}, ..., x_d) = u(t, x_1, ..., x_l, x_{l+1}, ..., x_d),$$
$$\text{and } u(t, x_1, ..., x_l, x_{l+1}, ..., x_d) = u_{\text{border}} \quad \text{with } x_j \in \{a_j, b_j\}, \text{and for} j = l+1, ..., d.$$

The function $H : [0, T] \times \Omega \times \mathbb{R}^d \to \mathbb{R}$ can be defined by:

- an analytic expression (if such an expression is known),

- a Hamiltonian corresponding to a one-player control problem:

$$H(t, x, p) := \max_{a \in \mathcal{A}} \Big(- f(t, x, a) \cdot \nabla u - \ell(t, x, a) \Big), \tag{1.3a}$$

or

$$H(t, x, p) := \min_{a \in \mathcal{A}} \Big(- f(t, x, a) \cdot \nabla u - \ell(t, x, a) \Big), \tag{1.3b}$$

where A is a set of control values, of the form $\prod_{i=1}^{m_A} [\alpha_i, \beta_i]$ (with $m_A \geq 1$), and where the drift $f : [0, T] \times \mathbb{R}^d \times \mathbb{R}^{m_A} \times \mathbb{R}^d$ and the distributed cost $\ell : [0, T] \times \mathbb{R}^d \times \mathbb{R}^{m_A} \to \mathbb{R}$ are given functions.

- a Hamiltonian function associated to a two-players game:

$$H(t, x, \nabla u) := \max_{a \in \mathcal{A}} \min_{b \in \mathcal{B}} \Big(- f(t, x, a, b) \cdot \nabla u - \ell(t, x, a, b) \Big), \tag{1.3c}$$

or

$$H(t, x, \nabla u) := \min_{a \in \mathcal{A}} \max_{b \in \mathcal{B}} \Big(- f(t, x, a, b) \cdot \nabla u - \ell(t, x, a, b) \Big), \tag{1.3d}$$

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

Steady equations The problem is to approximate $u = u(x)$ solution of

$$\begin{aligned} &\lambda u + H(x, \nabla u) = 0, \quad x \in \Omega \backslash \mathcal{C}, \\ &u(x) = 0, \quad x \in \partial \mathcal{C}, \\ &+ \text{ Boundary conditions on } \partial\Omega, \end{aligned}$$

with H is defined as in (1.3), Ω is a square of $\mathbb{R}^d$ of the form $\prod_{i=1}^{d} [a_i, b_i]$, the boundary conditions on $\partial\Omega$ are as in (1.2), and $\mathcal{C}$ is a given closed subset of Ω (it could be empty). The set $\mathcal{C}$ has to be defined by means of a function u_0, such that $\mathcal{C} := \{x,\ u_0(x) \leq 0\}$.

Time-dependent obstacle problem

$$\begin{cases} \min\left(\frac{\partial u}{\partial t} + H(t,x,\nabla u), u(t,x) - g_{\text{obs}}(t,x)\right) = 0, \quad x \in \Omega, \quad t \in [0,T], \\ u(0,x) = u_0(x), \quad x \in \Omega \\ + \text{ Boundary conditions on } \partial\Omega \times (0,T). \end{cases} \tag{1.4}$$

Here H, Ω and u_0 are as in the previous cases. The function $g_{\text{obs}} : (0,T) \times \mathbb{R}^d \to \mathbb{R}$ is a given function (in optimal control theory, this function represents obstacles or time-space regions that the trajectories should avoid, see [1]).

Steady obstacle equations

$$\begin{aligned} &\min\left(\lambda u + H(x,\nabla u), u(t,x) - g_{\text{obs}}(x)\right) = 0, \quad x \in \Omega\backslash\mathcal{C}, \\ &u(x) = 0, \quad x \in \partial\mathcal{C}, \\ &+ \text{ Boundary conditions on } \partial\Omega, \end{aligned}$$

The current version is compiled for a sequential execution, on a single computer. A parallel versions of the same library can be obtained upon request[1]

The library is distributed for three plate forms: Windows, Linux, Mac OS.

2 Implemented numerical methods.

The distributed version of the library. The version ROCHJ-v1.0 contains a static library `libhj.a` that implements the following classes of numerical methods:

- Finite difference methods: in these methods the discretization with respect to the time variable is performed by Euler scheme or Runge-Kutta method (RK2 or RK3). The discretisation in space is based on upwind finite difference, Lax-Freidrich method, or ENO2.
- Semi-Lagrangian methods: In this class of methods the user may choose an integration scheme for the characteristics. So far, the code includes several options: explicit Euler scheme, RK2, RK3, adaptif method using a number of intermediary time steps (this number can be fixed by the user). As for the interpolation method, the code uses only the bilinear scheme (other options will be implemented in the next version).

When the exact solution is known, the code can compute the numerical error in L^∞, L^2 or L^1 norms. The error can be estimated in all the domain Ω or just in a domain specified by the user (for instance, around the 0-level sets).

After the solution of the HJB equation is computed, a Matlab code can be run to plot this solution at different times steps. If the dimension of space variable exceeds $d \geq 3$ then the user may choose to plot a graph's cut with respect to some specific axis.

[1] Please send an email to `boka@math.jussieu.fr` or `Hasnaa.Zidani@ensta-paristech.fr`.

3 System requirements and basic usage

The software *ROC-HJ* contains:

- a pre-compiled library `libhj.a` (with the set of its header files)
- The main source file `main.cpp`
- A set of "model-description" files `data_XXX.h`

The library `libhj` contains the implemented numerical methods (classes `HJB, HJB_SL` and `HJB_FD`), all data management methods (classes RegularMesh and ParallelMesh). The figure 1 shows the structure of the library `libhj`.

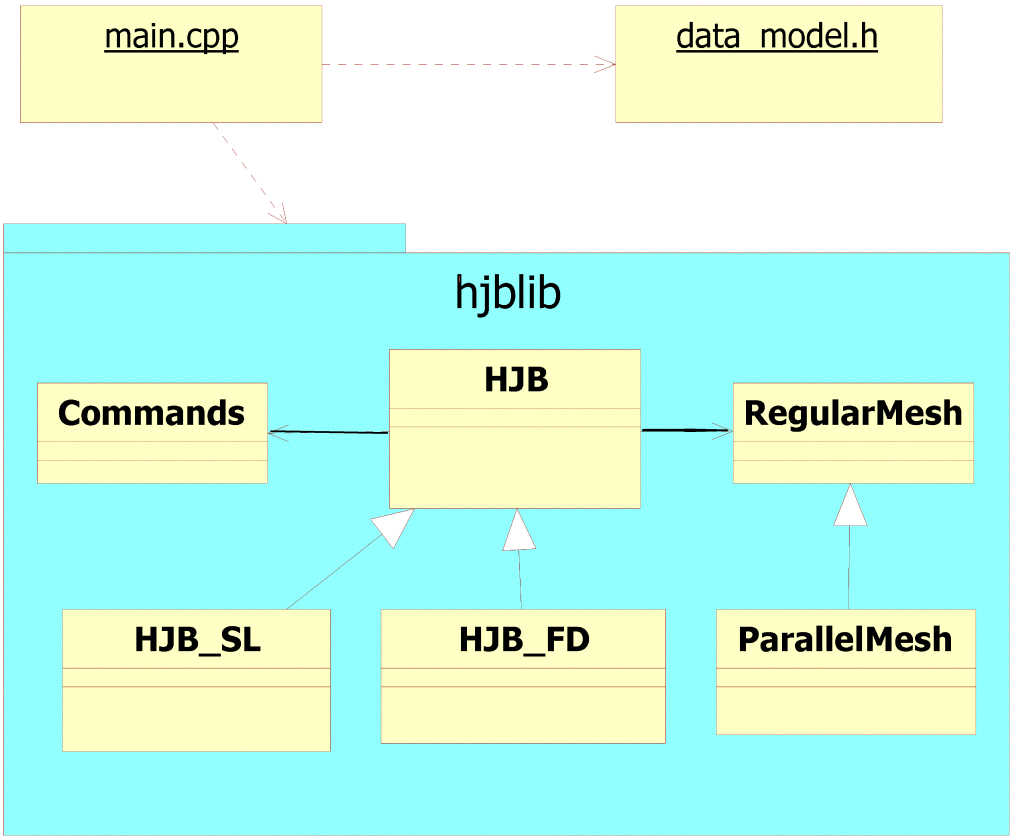


Figure 1: Library's architecture

The library was compiled and tested on three target systems: Windows, Linux and Mac OS. In all cases to use the library one needs to have some building tools for C/C++ installed on the computer:

- CMake Build chain (available for all systems at http://www.cmake.org/.)

- GCC compiler (native for Linux and Mac OS, MINGW distribution for Windows : http://www.mingw.org/

Some basic knowledge of C programming syntax is necessary to describe a new problem to be solved by the HJB-Solver. The user has to create a new model file as a C-header file, named like `data_yourModel.h`, that contains the definition of the drift, distributed-cost, the hamiltonian function (if known analytically), the initial conditions, the boundary conditions, obstacle function (if any) and all the parameters that describe the model to be solved.

The structure of data file is predefined and for the user's convenience several models of such a file are already available in the folder `Data`. In this folder two models are also provided:

- `data-BasicModel.h`: this file indicated the main functions that have to be implemented. It gives some basic options for the computation and for the plot of the results.
- `data-AdvancedModel.h`: here more options are indicated for the computation, error estimations, and the plot of results.

When the data file "`data_yourModel.h`" is created, the user should include its name in the beginning of the file `src/data_simulation.h`:

`\#include data_yourModel.h`

Then the project can be compiled and executed. Further detailed instructions on how to construct a model description header file and how to compile the programme are presented in the User's Manual for ROC-HJ solver.

4 Some examples

To illustrate different numerical methods implemented in the HJB Solver library some control problems were modeled. For each example model a Graphical User's Interface is proposed. It allows to define the parameters of each model and of the numerical method used to solve it. For time optimal control problems it is possible to compute the optimal trajectories for different initial conditions.

The graphical interfaces are developed with MATLAB and require a MATLAB licence to run.

4.1 Dubin's car example

We consider the navigation problem for a ground vehicle that has the objective of reaching a target in a given, finite time. The state of the system is characterized by the state vector (x, y, θ) where (x, y) are the 2D-coordinates of the center of mass of the vehicle in a given reference frame, and θ is the angle between the velocity vector and the x-axis. The state space is $\mathbb{R} \times \mathbb{R} \times [0, 2\pi]$. The motion of the vehicle obeys the following :

$$\begin{cases} x' = u\cos(\theta) \\ y' = u\sin(\theta) \\ \theta' = \omega \end{cases} \tag{4.5}$$

where the velocity module u is constant and the control input $\omega \in [\omega_{min}, \omega_{max}]$ is the angular speed.

The figure 2 shows the interface for the Dubin's Car model. It is possible to fix the time horizon, T, the vehicle's velocity u, the position and the shape of the target set (disc or rectangle).

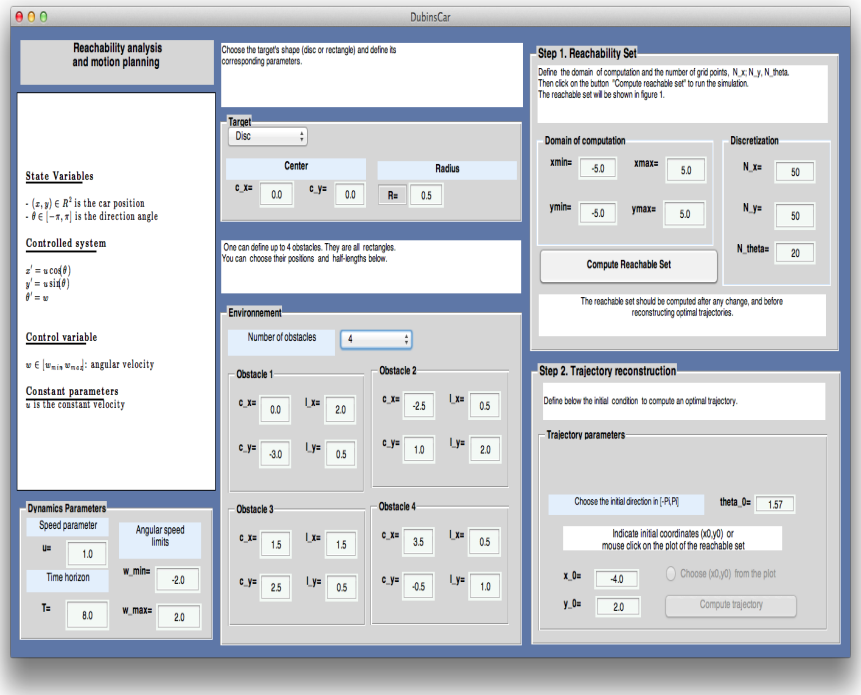
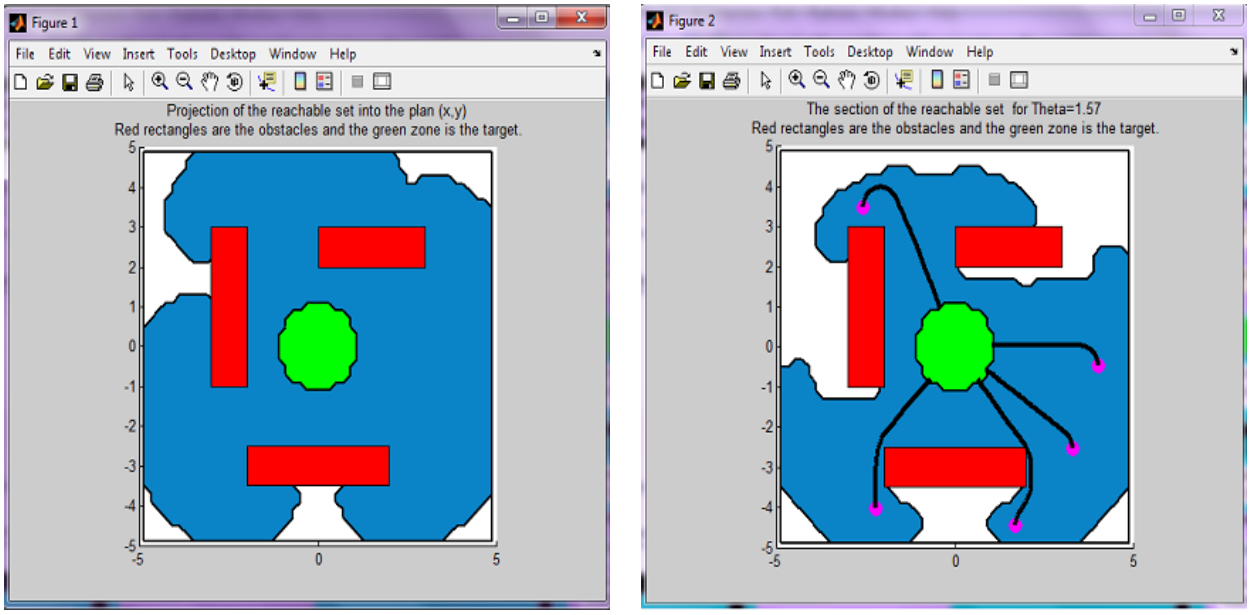


Figure 2: Graphical interface for the Dubin's car model



(a) (b)

Figure 3: (a) Example of reachable set; (b) Some optimal trajectories

On can also choose up to 4 obstacles that the car has to avoid during its trajectory to reach the target. These obstacles are considered as state constraints on the variables (x, y). The first step of the simulation computes and plots the reachable set corresponding to the fixed parameters, as on the figure 3-(a). After the computation of the reachable set it is possible to reconstruct optimal minimum time trajectories by choosing different initial conditions (see figure **??**-(b)).

4.2 Zermelo's navigation example

In this example, we consider a ship moving with contant relative velocity in a river of a given width. We assume also that the current's velocity is known, and the state of the system is the position (x, y) of the ship. The motion is given by a nonlinear system in $\mathbb{R}^2$ controlled by the steering direction $\theta \in [0, 2\pi]$. The evolution of the swimmer is governed by the control system

$$F : \begin{cases} x' = c - ay^2 + u\cos(\theta) \\ y' = u\sin(\theta) \end{cases} \tag{4.6}$$

The ship aims to join an island (disc of radius R, where the parameter R can be chosen by the user).

The figure 4 shows the interface for this model. It is possible to fix the time horizon, T, the ship's velocity, u, and the position of the target set (disc). The user can also choose up to 2 obstacles in the environnement. In this example, the first step of the simulation computes and plots the back-ward reachable set within time T, as on the figure 5. After the computation of the reachable set it is possible to reconstruct optimal minimum time trajectories by choosing different initial conditions (see figure 6).

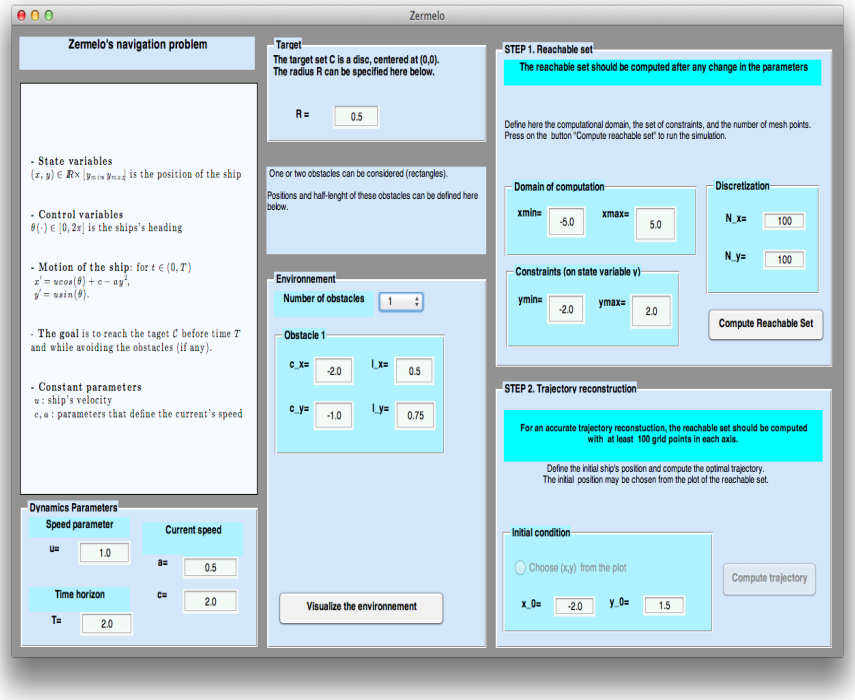


Figure 4: Graphical interface for Zermelo's navigation model

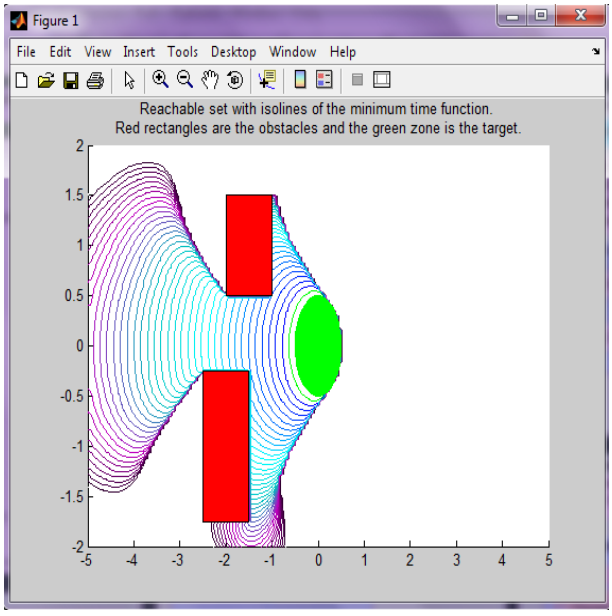


Figure 5: Example of backward reachable set for Zermelo's navigation problem

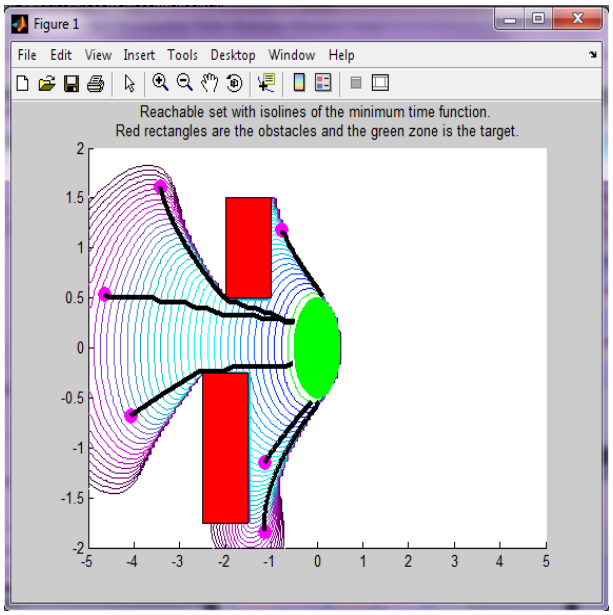


Figure 6: Some optimal trajectories for Zermelo's problem

4.3 Advection example

This interface allows to compare four numerical methods for solving a linear HJ-equation (Advection-rotation example):

$$\frac{\partial v}{\partial t} + H(X, D_X v) = 0, \quad \text{with } v(X,0) = v_0(X),$$

with the Hamiltonian defined by:

$$H(X, D_X v) = (-f(X_1, X_2), D_X v)$$

whith $f : R^2 \to R^2$ is given by:

$$f(X_1, X_2) = \begin{pmatrix} -2\pi X_2 \\ 2\pi X_1 \end{pmatrix}.$$

In this example, we compare the methods:

- finite difference scheme, with Lax-Freidrich approximation of the Hamiltonian
- finite difference scheme, with ENO2 approximation of the Hamiltonian
- Semi-lagrangian scheme with RK2 scheme for the integration of charscteristics
- Semi-Lagrangian scheme with exact integration of the characteristics

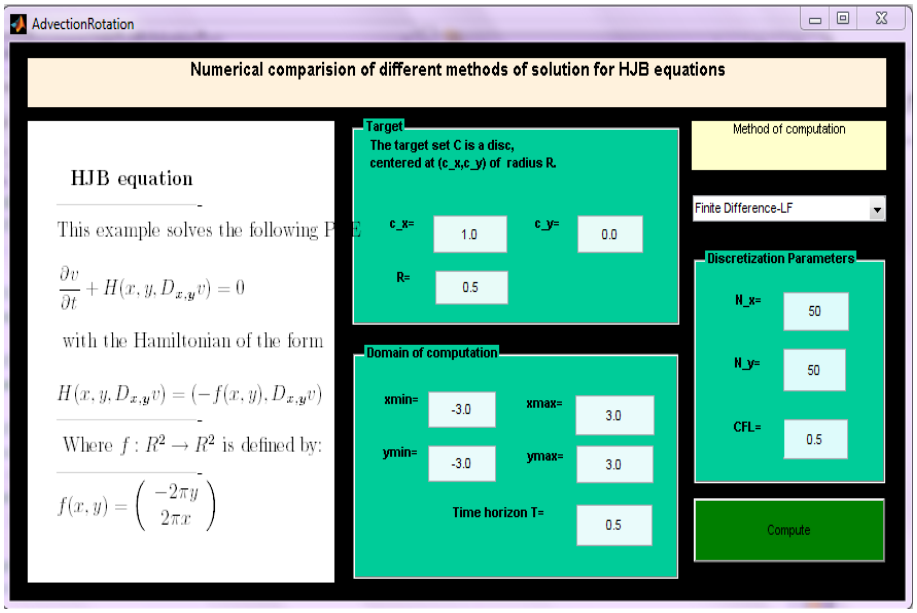


Figure 7: Graphical interface for the Advection model

The figure 9 shows the interface for this problem. It is possible to fix the time horizon, T, the radius of the target set (disc).

During the simulation, some local error estimates (in L^∞, L^1, and L^2) are printed on the Matlab workspace.

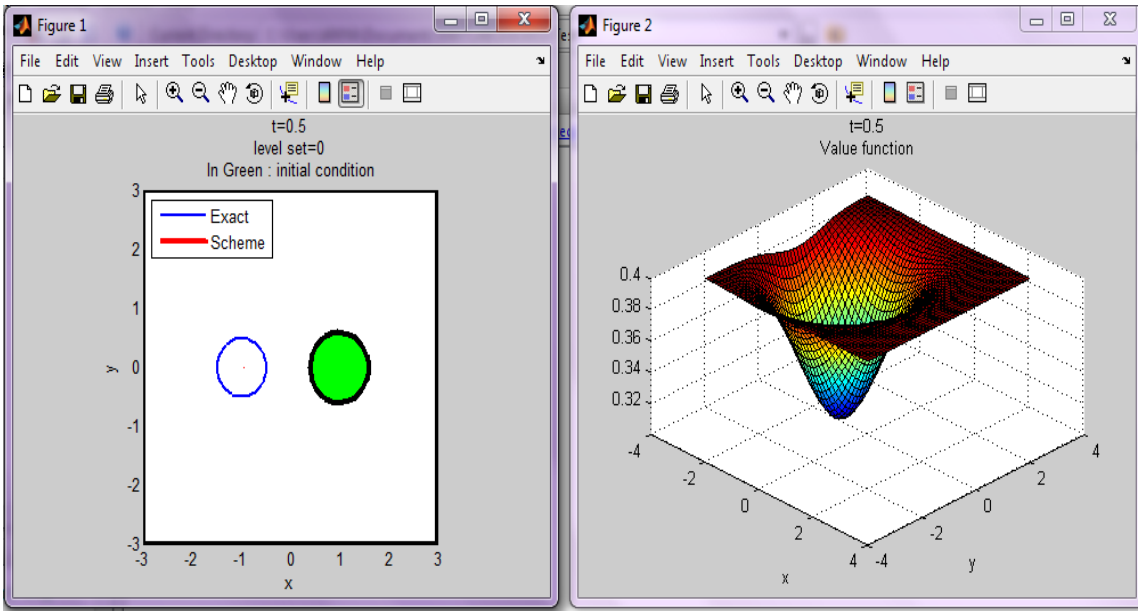


Figure 8: Example of the results

4.4 Collision avoidance for a UAV

In this example, we consider a collision avoidance problem. Here the ROC-HJ software is used to compute nested safety regions around a UAV with the goal to analyze if a collision can occur or not. Moreover, the software can compute the optimal strategy to avoid the collision when it cannot be avoided without maneuver. Details on this example are given in [4].

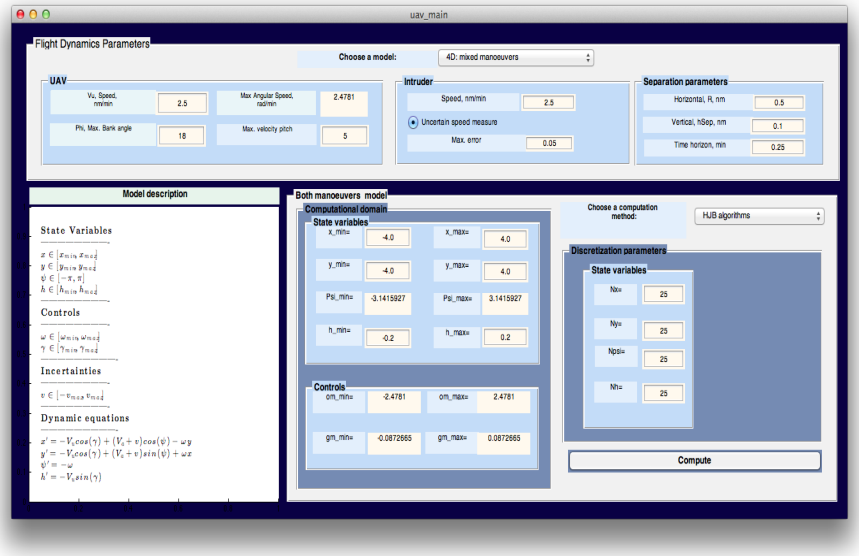


Figure 9: Graphical interface for "Collision avoidance for UaVs"

References

[1] A. Altarovici, O. Bokanowski, and H. Zidani. A general Hamilton-Jacobi framework for nonlinear state-constrained control problems. *ESAIM: Control, Optimisation and Calculus of Variations*, 2012.

[2] O. Bokanowski, N. Forcadel, and H. Zidani. Reachability and minimal times for state constrained nonlinear problems without any controllability assumption. *SIAM J. Control and Optimization*, 48(7):4292–4316, 2010.

[3] O. Bokanowski and H. Zidani. Minimal Time Problems with Moving Targets and Obstacles. In *18th IFAC World Congress*, volume 18, Part 1, pages 2589–2593, Milano, Italie, 2011.

[4] E. Crück, A. Desilles, and H. Zidani. Collision analysis for a UAV. In *Proceedings of AIAA Guidance, Navigation, and Control conference*, pages AIAA 2012–4526, Minneapolis, États-Unis, Aug. 2012.

[5] M. Falcone and R. Ferretti. Convergence analysis for a class of high-order semi-Lagrangian advection schemes. *SIAM J. Numer. Anal.*, 35(3):909–940 (electronic), 1998.

[6] S. Osher and C.-W. Shu. High essentially nonoscillatory schemes for Hamilton-Jacobi equations. *SIAM J. Numer. Anal.*, 28(4):907–922, 1991.