RaBVItG: An Algorithm for Solving a Class of Multi-Players Feedback Nash Differential Games

Jorge Herrera de la Cruz[a], Benjamin Ivorra[b], Ángel M. Ramos[b]

[a] Department of Economic Analysis and Quantitative Economics
Complutense University of Madrid
Campus de Somosaguas, s/n, 28223 Pozuelo de Alarcón, Spain
email: joherrer@est-econ.uc3m.es

[b]Instituto de Matemática Interdisciplinar (IMI),
Department of Mathematical Analysis and Applied Mathematics,
Complutense University of Madrid,
Plaza de las Ciencias, 3, Madrid 28040, Spain
email: ivorra@ucm.es;angel@ucm.es

April 12, 2019

Abstract

In this work, we introduce a novel numerical algorithm, called RaBVItG (Radial Basis Value Iteration Game) to approximate feedback-Nash equilibria for deterministic differential games. More precisely, RaBVItG is an algorithm based on value iteration schemes in a meshfree context. It is used to approximate optimal feedback Nash policies for multi-players, trying to tackle the dimensionality that involves, in general, this type of problems. Moreover, RaBVItG also implements a game iteration structure that computes the game equilibrium at every value iteration step, in order to increase the accuracy of the solutions. Finally, with the purpose of validating our method, we apply this algorithm to a set of benchmark problems and compare the obtained results with the ones returned by another algorithm found in the literature. When comparing the numerical solutions, we observe that our algorithm is less computationally expensive and, in general, reports lower errors.
Keywords: Differential Games, Feedback Nash Equilibrium, Hamilton-Jacobi-Bellman equation

1 Introduction

From a general point of view, differential games (DG) is a topic involving game theory and controlled systems of differential equations. Here, we focus on conflict problems in which N players control a system with an associated cost function per player. There exist two main types of control systems for these problems: *closed-loop* (or feedback), in which the controls (optimal solutions) are functions of the state variables of the system; and *open-loop* solutions, in which controls are only functions of time. Focusing on differential games, feedback controls are more robust than open-loop controls due to the possibility of the players to react to sudden changes in the values of the state variables of their opponents while they are playing the game. Based on the behavior of the N players, it is possible to classify the game and its corresponding equilibrium solutions as *cooperative* and *non cooperative.* The first class is used when the N players make coordinated decisions trying to optimize a join criterium,

while the second class is used when the N players compete each other trying to optimize their own criterium, taking into account the strategies from the rest $N-1$ players. We note that there also exist intermediate possibilities.

In this work, we focus on Nash (i.e., *non cooperative)* feedback solutions (i.e., *closed-loop).* Our aim is twofold. Firstly, we develop an algorithm to solve numerically a deterministic multi-players Feedback Nash Differential Game (FNDG). The algorithm is called RaBVIt (Radial Basis Value Iteration Game), and is based on: radial basis, value iteration and game iteration. Secondly, we apply the algorithm to some benchmark problems found in the literature of differential games (generally, linear or linear quadratic examples with explicit solutions) in order to compare the performance of RaBVItG to a previous algorithm (see [5], for more details).

Focusing on recent works, we highlight the fact that in a recent survey (see, [13]), the authors report that some existing numerical (computational) methods in the literature of DG are focused in some feasible subclass of games: two-person zero-sum games (see, [9]). These games, do not strictly match our purpose (of dealing with N players non-zero games) but are interesting , from an Analytical point of view, because they tackle the lack of differentiability of the value function by using the viscosity solution, as this scheme can be reduced to a one dimensional game. Viscosity solutions in N dimensional games are, in general, complex to obtain. Additionally, other advances are in the line of Linear Quadratic models (see, [8]), but this frame-work omits to deal with interesting non linearities in the dynamic system or running cost functions. According to the authors, computational differential games is an area that needs to grow up.

In this context, RaBVItG is an algorithm based on some recent reinforcement learning schemes (see, e.g., [15]) which are discrete-time methods closely related to dynamic programming. More precisely, we consider methods from reinforcement learning algorithms that are, mainly, used to simulate and approximate the behavior (in their usual terminology) of a set of *agents* that take *actions* in order to get a cumulative *reward*. In particular, we focus on value iteration (VI) which is a general technique used by reinforcement learning to iterate over the value function of the problem in order to obtain a fixed point solution. In our case, we have a coupled system of value functions (one per player) involved in a FNDG. We also use in RaBVItG the concept of game iteration (GI), meaning that at each *value iteration* step, we iterate again to find the corresponding game equilibrium (Nash, in this case) associated to the set of value functions. This algorithm is used to simulate the way in which the players make decisions (for instance, in a Nash context, fixing the opponent's strategies to obtain the optimal strategy for a player). Finally, we also use function approximation techniques that allow us to simplify the model in order to approximate, using mesh-free techniques, the value function for each player (see for instance, [**?**, 16]).

This numerical method has been designed for solving a coupled system of N-player Hamilton-Jacobi-Bellman equations (HJB). HJB equations provide the value function which gives optimal policies for a given dynamic system with an associated cost function (see, for example, [7]. They are the key for finding a feedback solution. In order to discretize our problem (in time and space), we use the techniques developed in [10], which introduces a semi-lagrangian discretization framework for HJB equations and proves the convergence of the scheme based on the viscosity solution (see, e.g, [6]).

In order to validate our algorithm, we apply it to solve a set of benchmark problems found in the literature (see, [8, 5]). We compare the obtained CPU time and error with the ones returned by another numerical algorithm found in [5].

This paper is organized as follows. In Section 2, we present the theoretical model by describing the relevant variables involved in the game, the coupled optimization problems and the basic Nash equilibrium concepts. In Section 3, we explain the numerical implementation of our method, based on a semi-lagrangian discretization, value iteration and Radial Basis interpolation. In Section 4, we show the performance of the method by solving some benchmark problems.

2 Materials and Methods

This section deals with the explanation of the considered deterministic theoretical model. Firstly, we introduce the differential game of interest. Secondly, we define the considered feedback Nash equilibrium and the Hamilton-Jacobi-Bellman equations.

2.1 Deterministic Differential Game

We first define the class of differential games we are dealing with in this work.

Let us consider a set of N players. Each player, $i \in \{1, 2, ..., N\}$, has a payoff functional, $J_i(y, u)$ given by:

$$J_i(y,u) = \int_0^\infty e^{-\rho_i t} f_i(x(t), u(t))\, \mathrm{d}t, \tag{1}$$

with $y \in X$, $u \in \mathcal{U}$ and

$$\dot{x}(t) = g(x(t), u(t)) \;\; t \in [0, \infty), \tag{2}$$

and

$$x(0) = y. \tag{3}$$

defining

1. $u : [0, \infty) \to U$, is the control function with $U \subset \mathbb{R}^m$. In our case, $U = \bigcup_{i=1}^N U_i$, so that we can define the same function for each player. Let us to define $u_i : [0, \infty) \to U_i$, as the control associated to the i- th player, with $U_i \subset \mathbb{R}^{m_i}$ a given subset of admissible controls. We also denote by $\mathcal{U}_i = \{u_i : [0, \infty) \to U_i, \text{such that } u_i \text{ is measurable}\}$, $\mathcal{U} = \bigcup_{i=1}^N \mathcal{U}_i$ and $u(t) = (u_1(t), ..., u_N(t)) \in \mathcal{U}$.

2. $x : [0, \infty) \to X$ is a function such that $x(t)$ denotes the state of the system at time t, where $X \subset \mathbb{R}^P$ is the set of admissible states ($p \neq N$). The evolution of those state variables is driven by (2), called the *state equation.* We note that $y \in X$ and $g : D \to \mathbb{R}^P$, with $D \subset \mathbb{R}^P \times \mathbb{R}^m$ and assume that g is continuous and it exists L_g such that, $|g(x_1, u) - g(x_2, u)| \leq L_g |x_1 - x_2|$ for all $x_1, x_2 \in X$, $t \in \mathbb{R}^+$ and $u \in U$. Those conditions ensure that the system has an unique solution (can be proved by using the Carathèodory theorem, see [10]).

3. $J_i : X \times \mathcal{U} \to \mathbb{R}$ is the payoff functional of player i . We assume $f_i : D \to \mathbb{R}$, where $f_i(x(t), u(t))$ represents the instantaneous payoff of player i for the choice $u_i(t)$. Note that the integral is affected by an exponential discounting parameter $\rho_i > 0$ that actualizes the value of the payoff (see, for instance [8]). The presence of the discount factor ensures that the integral (1) is finite whenever f_i is bounded.

2.2 Feedback Optimization Problem

In this Section, we recall the *synthesis procedure* detailed in, e.g [1].

Let $v_i : X \to \mathbb{R}$, be a continuously differentiable mapping called *value function* and defined by:

$$v_i(y) = \sup_{u \in \mathcal{U}} J_i(y, u).$$

For each $y \in X$, we assume the existence of, at least, one optimal control $u_y^* \in \mathcal{U}$ such that

$$v_i(y) = \max_{u \in \mathcal{U}} J_i(y, u) = J_i\left(y, u_y^*\right) = \int_0^\infty e^{-\rho_i t} f_i\left(x_y\left(t, u_y^*\right), u_y^*(t)\right) \mathrm{d}t, \tag{4}$$

where $x_y(t, u_y)$ is an admissible trajectory satisfying (2)-(3). We denote by $x_y^*(t) = x_y(t, u_y^*)$ the optimal trajectory.

According to [1], the function $h_{y,u_y}^i \; [0, \infty) \to \mathbb{R}$ given by

$$h_{y,u_y}^i(\tau) := \int_0^\tau \left(f_i(x_y(t; u_y), u_y(t))\, e^{-\rho_i t} \mathrm{d}t + v_i\left(x_y(\tau; u_y)\right) e^{-\rho_i \tau}\right.$$

is constant and non increasing for all $t > 0$ if and only $u_y \in \mathcal{U}$ is an optimal control for the initial condition y. Thus, considering $u_y^* \in \mathcal{U}$, and derivating h^i_{y,u_y} with respect to τ , we obtain

$$\rho_i v_i\left(x_y^*(\tau)\right) - f_i(x_y^*(\tau), u_y^*(\tau)) - \nabla v_i(x_y^*(\tau))^T g\left(x_y^*(\tau), u_y^*(\tau)\right) = 0.$$

Furthermore, for $\tau = 0$, we have that

$$\rho_i v_i(y) - f_i(y, u_y^*(0)) - \nabla v_i(y)^T g\left(y, u_y^*(0)\right) = 0.$$

We note that for any other admissible constant control of the form $u(t) = \xi \in U$, for all $t \geq 0$, as$h^i_{y,u_y}(\tau)$ is not increasing, we have that

$$\frac{\mathrm{d}}{\mathrm{d}\tau}\left[\int_0^\tau f_i\left(x_y(\tau), \xi\right) e^{-\rho_i t}\mathrm{d}t + v\left(x_y(\tau, \xi)\right) e^{-\rho_t \tau}\right] \leq 0.$$

Again, for $\tau = 0$, we obtain

$$\rho_i v_i(y) - f_i(y, \xi) - \nabla v_i(y)^T g\left(y, \xi\right) \leq 0.$$

So, under previous hypothesis, we conclude that for all $y \in X$ we satisfy the following HJB equation

$$\rho_i v_i\left(y\right) = \max_{u \in U}\left\{f_i\left(y, u\right) + \nabla v_i\left(y\right)^T g(y, u)\right\}, \tag{5}$$

and u_y^* is such that

$$u_y^*(0) \in \operatorname*{argmax}_{\xi \in U}\left\{f_i\left(y, \xi\right) + \nabla v_i\left(y\right)^T g(y, \xi)\right\}. \tag{6}$$

We remark that, in general, if v_i is only continuous, such equation needs to be interpreted in terms of viscosity solution of Problem (5). However, when it applies to differential games, there are, so far, no general theorem on the existence or uniqueness of solutions (see, [3]).

Now, we define $S_i : X \to U_i$ called *feedback-map* per player $i = 1, ..., N$, such that

$$S_i(y) \in \arg\max_{u \in U}\left\{f_i(y, u) + \nabla V_i(y)^T g\left(y, u\right)\right\}, \tag{7}$$

where $y \in \mathbb{R}^p$. We consider an optimal control defined by

$$u_y^*(t) = S_i(x_y^*(t)), \tag{8}$$

for almost every $t > 0$.

The above mentioned synthesis procedure consists in obtaining, using the *feedback-map,* an optimal decision $u_y^*(t) = S_i(x_y^*(t))$ related to the corresponding optimal trajectory, by solving

$$\dot{x}\left(t\right) = g\left(x(t), \left[S_1\left(x(t)\right), ..., S_N\left(x(t)\right)\right]\right) \; t \in [0, \infty), \tag{9}$$

and

$$x(0) = y. \tag{10}$$

So, provided an initial position y, we say that $\left(u_y^*, x_y^*\right) \in \mathcal{U} \times C\left([0, \infty), X\right)$ is an optimal pair *control-trajectory* for every initial condition, and it corresponds to the optimal feedback policy we are trying to estimate.

2.3 Feedback - Nash Equilibrium

Now, we adapt the previous expressions in order to get a feedback-Nash equilibrium.

To do so, we define a *feedback-Nash map* $S_i : X \to U_i$, per player $i = 1, ..., N$, such that for all $y \in \mathbb{R}^p$,

$$S_i(y) \in \arg\max_{u_i \in U_i} \left\{ f_i(y, [u_i; u_{-i}]) + \nabla V_i(y)^T g\left(y, [u_i; u_{-i}]\right) \right\}, \tag{11}$$

where the array $[u_i; u_{-i}]$ denotes the pair of the control for player i and the controls associated to the rest of players (denoted by $-i$).

Considering this feedback-Nash map, we apply the synthesis procedure described previously. To do so, we define a feedback N-tuple $S \equiv (S_1, ..., S_{i-1}, S_i, S_{i+1}, ..., S_N)$ and a feedback (N-1)-tuple $S_{-i} \equiv (S_1, ..., S_{i-1}, S_{i+1}, ..., S_N)$. Then,$S_* \equiv (S_{*,1}, ..., S_{*,i-1}, S_{*,i}, S_{*,i+1}, ..., S_{*,N})$ is a feedback- Nash equilibrium (FNE) if

$$J_i\left(y; S_*\right) \geq J_i\left(y; [u_i, S_{*,-i}]\right) \text{ for each } y \in X,\ S_i : X \to U_i \text{ and } i \in \{1, ..., N\}. \tag{12}$$

where $J_i(y, S_i)$ denotes $J_i(y, S_i(y))$, and $[u_i, S_{*,-i}]$ is the $N-$vector of controls obtained by replacing the i-th component in S_i^* by u_i.

Assuming y and $S_{*,-i}$ are fixed, finding u_i by maximizing $J_i\left(y; [u_i, S_{*.-i}]\right)$ (i.e, solving Equation (12)) can be solved by using dynamic programming (see, for instance, [12, 16]).

Finally, we define

$$v_i(y; S_{*,-i}) = \max_{u_i \in \mathcal{U}_i} J_i(y; [u_i, S_{*,-i}]), \tag{13}$$

which is solution of the following HJB equation (see [9]):

$$\rho_i v_i\left(y; S_{*,-i}\right) = \max_{u_i \in U_i} \left\{ f_i\left(y, [u_i S_{*,-i}]\right) + \nabla V_i(y; S_{*,-i})^T g\left(y, [u_i, S_{*,-i}]\right) \right\}. \tag{14}$$

where the notation $v_i\left(.; S_{*,-i}\right)$ means that we are finding a Nash equilibrium for player i, by fixing the optimal strategies of the $N-1$ players. We note that regarding current literature, apart from zero-sum games, to find relevant cases where (14) is well posed, we need to focus on games in one spatial dimension. For instance, in [4], an existence theorem of Nash equilibria in feedback form, valid for one-space dimension *non cooperative* games, is given. However, as far as we know, there is no general existence theorems for feedback-Nash Equilibrium (for spaces in dimension greater than 1).

3 The RaBVItG Algorithm for Solving Deterministic Differential Games

In this section, we describe the numerical implementation of the algorithm RaBVItG used to solve the considered deterministic differential game presented in Section 2.2. To do so, we propose a semi-lagrangian discretization scheme of the HJB equation (see [10]). Then, we describe the general structure of the algorithm used to solve this problem. Finally, we introduce a particular implementation for the case of a feedback-Nash N-players differential game.

3.1 Model Discretization

Here, we propose a particular discrete version of equations (1)-(3).

Let $h \in (0, 1)$ be a time step and $t_k = kh$, $k \in \mathbb{N} \cup \{0\}$. First, we aim to approximate $x_k \approx x(t_k)$. To do so, given $y \in X$, we consider the following discrete approximation of (1)-(2):

$$J_i^h\left(y, u^h\right) = h \sum_{k=0}^{\infty} e^{-\rho_i k} f_i(x_k, u_k), \tag{15}$$

where $u_k = [u_{1,k}, ..., u_{N,k}] \in U$, with $u_{i,k} \in U_i$, being $U = \bigcup_{i=1}^{N} U_i$ and with the assumption that the controls are constant on $[t_k, t_{k+1})$; $u^h = \left[u_1^h, ...u_N^h\right] \in \mathcal{U}$, with $u_i^h \in \mathcal{U}_i^h$ the control for player i, and $\mathcal{U}^h = \bigcup_{i=1}^{N} \mathcal{U}_i^h$, being $\mathcal{U}_i^h = \left\{u_i^h \,:\, u_i^h = \{u_{i,0}, u_{i,1}, ...\}\right.$ with $\left. u_{i,k} \in U_i\right\}$.

Then, starting from $x_0 = y$, we use a first-order Euler scheme for the state equations

$$x_{k+1} = x_k + hg(x_k, u_k) \tag{16}$$

Next, we discretize the HJB equation (14). To this aim, given $y \in \mathbb{R}^N$, we approximate Equation (13) by considering

$$v_i^h(y) = \max_{u_i^h \in \mathcal{U}_i^h} J_i^h\left(y, \left[u_i^h, u_{-i}^h\right]\right), \tag{17}$$

Following [10], we obtain a first order discrete-time HJB equation for player i (i.e., a discrete version of (14)) of the form:

$$v_i^h(y) = \max_{u_i^h \in U_i} \left\{hf_i\left(y, \left[u_i^h, u_{-i}^h\right]\right) + (1 - \rho_i h) v_i^h\left(y + hg\left(y, \left[u_i^h, u_{-i}^h\right]\right)\right)\right\}, \tag{18}$$

we point out that $1 - \rho_i h$ corresponds to the first order Taylor expansion of $e^{-\rho_i h}$ which is typically used as a discounting factor in discrete dynamic programming (see, i.e., [2]).

Now, focusing on the synthesis procedure, we define the following discrete-time versions of *discrete feedback N-tuple as* $S^h \equiv \left(S_1^h, ..., S_{i-1}^h, S_i^h, S_{i+1}^h, ..., S_N^h\right)$ and the *discrete feedback (N-1)-tuple as* $S_{-i}^h \equiv \left(S_1^h, ..., S_{i-1}^h, S_{i+1}^h, ..., S_N^h\right)$. Note that the *discrete feedback-Nash* $S_*^h \equiv \left(S_{*1}^h, ..., S_{*N}^h\right)$ satisfies

$$J_i^h\left(y; S_*^h\right) \geq J_i\left(y; \left[u_i^h, S_{*,-i}^h\right]\right), \tag{19}$$

for each $y \in X$, $S_i^h \,:\, X \to U_i$ and $i \in \{1, ..., N\}$.

Furthermore, according to the definition of the *feedback-map,*

$$S_i^h(y; S_{-i}^h) \in \arg\max_{u_i^h \in U_i} \left\{hf_i(y, \left[u_i^h, S_{-i}^h\right]) + (1 - \rho_i h) v_i^h\left(z + hg\left(y, \left[u_i^h, S_{-i}^h\right]\right); S_{-i}^h\right)\right\}. \tag{20}$$

Thus, we obtain

$$v_i^h(y) = h \sum_{k=0}^{\infty} e^{-\rho_i k} f_i(x_k^*, S^h(x_k^*)), \tag{21}$$

where, $x_0^* = y$ for $k \in \mathbb{N}$ and

$$x_{k+1}^* = x_k^* + h \cdot \left(x_k^*, S^h(x_k^*)\right). \tag{22}$$

However, to determine v_i^h satisfying (18) is still not always feasible. Thus, we aim to obtain an approximation, denoted by $v_{i,j}^h$, by considering a spatial discretization of (14). To do so, we consider a set of $Q \in \mathbb{N}$ arbitrary points $\{y_j\}_{j=1}^{Q} \in \widetilde{X}$, with $\widetilde{X}$ being a closed subset of X. Next, we approximate $v_{i,j}^h\left(y_j; S_{*-i}^h\right)$ for each $i \in \{1, ..., N\}$ and $j \in \{1, ..., Q\}$. To this aim, let $y^\sharp = \left(y_j + h \cdot g\left(y_j, \left[u_i, S_{*-i}^h\right]\right)\right) \in \widetilde{X}$ (which are not necessarily in $\widetilde{X}$). For those points, the HJB equation (18) can be approximated as

$$v_{i,j}^h\left(y_j; S_{*-i}^h\right) = \max_{u_i \in U_i} \left\{hf_i\left(y_j, \left[u_i, S_{*-i}^h\right]\right) + (1 - \rho_i h) I\left[V_i\right]\left(y^\sharp; S_{*-i}^h\right)\right\}, \tag{23}$$

where $I\left[V_i\right](\cdot)$ is the approximation of $v_{i,j}^h\left(y^\sharp; S_{*-i}^h\right)$ at points $y^\sharp$ computed by a collocation algorithm (see, for instance, [**?**]) using a "mesh-free" method based on scattered nodes.

More precisely, $I\left[V_i\right](\cdot)$, is of the form

$$I\left[V_i\right]\left(y^\sharp\right) = \sum_{j=1}^{P} \lambda_j \Phi\left(\left\|y^\sharp - y_j\right\|_2\right),\ y^\sharp \in \widetilde{X} \tag{24}$$

where $\lambda_j \in \mathbb{R}$, $\|\Delta\|_2$ is the Euclidean norm and $\Phi(r)$ is a real-valued radial basis function (see, for instance, [11]). Here, we use the Gaussian RBF given by $\Phi(r) = \exp\left(-\frac{\left\|y^\sharp - y_j\right\|_2}{2\sigma^2}\right)$ with $\sigma^2 > 0$. In order to determine λ_j, for $j = 1, ..., Q$, we consider that

$$\mathrm{A}\bar{\lambda}_i = \mathrm{V}_i \tag{25}$$

Where $\mathrm{A}_{jl} = \Phi\left(\|y_j - y_l\|\right)$, for $j, l = 1, ..., Q$, $\bar{\lambda}_i = \left[\lambda_1, .., \lambda_Q\right]^T$ and $\mathrm{V}_i = \left[v_i^h\left(y_1\right), ..., v_i^h\left(y_Q\right)\right]^T$ for $i = 1, ..., N$.

3.2 Algorithm Structure

In this Section, we present the general structure of the algorithm used to solve the problem defined by Equations (1)-(3) and (12) and using the discrete HJB equation (23). This algorithm is based on two main nested loops. It combines a process of the main loop called n *value iteration* (see [12]) with an inner loop called *game iteration (GI)* , consisting in a relaxation algorithm to find the proper (convergent) Nash equilibrium for an *approximated* value (until convergence of this value).

Firstly, before presenting the algorithm, we introduce some useful notations. Let V be the array of values for all the players evaluated in all the original set of points $\widetilde{X}$ and given by

$$\mathrm{V} = \left[v_1^h\left(y_1\right), ..., v_1^h\left(y_Q\right), ..., v_N^h\left(y_1\right), ..., v_N^h\left(y_Q\right)\right]^T \in \mathbb{R}^{Q\cdot N}.$$

We also define Λ as a matrix that stores the controls of each player as follows

$$\mathbf{\Lambda} = \left(\Lambda_1, ..., \Lambda_i, ..., \Lambda_N\right) \in \mathcal{M}_{Q\cdot N\times m}$$

with $\Lambda_i = \left[\left(u_i\right)_1, ..., \left(u_i\right)_Q\right]^T \in \mathcal{M}_{Q\times m}$, and $\mathcal{M}_{n\times m}$ denotes the set of all real-valued matrices of dimension $n \times m$. We also define

$$\mathbf{\Lambda}_{-i} = \left(\Lambda_1, ..., \Lambda_{i-1}, \Lambda_{i+1} ..., \Lambda_N\right) \in \mathcal{M}_{Q(N-1)\times m}.$$

Additionally, let F be a vector that quantifies the cost for each player at every data points, given by

$$\mathrm{F}(\Lambda) = \left[\left(f_1\left(x, \Lambda\right)\right)_1, ..., \left(f_1\left(x, \Lambda\right)\right)_p, ..., \left(f_N\left(x, \Lambda\right)\right)_1, ..., \left(f_N\left(x, \Lambda\right)\right)_p\right]^T \in \mathbb{R}^{Q\cdot N},$$

Next, we introduce two operators, $T = \left(T_1, ..., T_{Q\cdot N}\right) : \mathbb{R}^{Q\cdot N} \to \mathbb{R}^{Q\cdot N}$, and $G = \left(G_1, ..., G_{Q\cdot N}\right) : \mathbb{R}^{Q\cdot N} \to \mathbb{R}^{Q\cdot N}$, with

$$T_j\left(\mathbf{V}; \mathbf{\Lambda}\right) = h\mathbf{F}(\Lambda)_j + \left(1 - \rho_i h\right)\left(I[\mathbf{V}]\left(\Lambda\right)\right)_j,\ j = 1, ..., Q\cdot N,$$

and

$$G_{i,j}\left(\Lambda_i; \mathbf{V}_i\right) = \arg\max_{\Lambda_i \in U_i}\left(J_i\left(y, \left[\Lambda_{i,j}, \Lambda_{-i,j}\right]\right)\right)\ i = 1, ..., N,\ j = 1, ..., Q\cdot N.$$

In the previous expression, $I[\mathbf{V}]\left(\Lambda\right) = \begin{pmatrix} \mathrm{A}\cdot\overline{\lambda_1} & ... & ... & \mathrm{A}\cdot\overline{\lambda_N} \end{pmatrix} \in \mathbb{R}^{Q\cdot N}$, is a $Q\cdot N$ interpolation block vector where $\overline{\lambda_i}$ and A defined in (25).

Secondly, we pretend to solve the following fixed-point schemes

$$\mathrm{V} = T(\mathrm{V}; \Lambda, \triangle) \text{ and } \Lambda = G(\Lambda; \mathrm{V}, \triangle),$$

where $\triangle$ denotes *spatio-temporal discretization parameters*. More precisely, let $s = 0, 1, ...$ and $r = 0, 1, ...$ We define the following processes:

- ***Game iteration:*** first, for each player i, we generate Λ_i^{s+1}, a candidate to optimal policy at step *s+1* as following

$$\left(\Lambda_i^{s+1}\right)_j = \theta\left(\Lambda_i^s\right)_j + (1-\theta)\arg\max_{\Lambda_i \in U_i}\left(h\mathrm{F}(\left[\Lambda_i, \Lambda_{i-1}^s\right])_j + (1-\rho_i h) I\left[\mathrm{V}_i^r\right]\left(\left[\Lambda_i, \Lambda_{i-1}^s\right]\right)_j\right) \; j = 1, ..., Q \cdot N \tag{26}$$

with $y^\sharp = y_j + hg(y_j, \left[\Lambda_i, \Lambda_{-i}^s\right]_j)$, $0 < \theta < 1$ a weight coefficient (see, e.g., [14]) and V_i^r described below.

We iterate this process until convergence (i.e., considering $\left\|\Lambda^{s+1} - \Lambda^s\right\| < \epsilon_1$, with $\epsilon_1 > 0$). Once the convergence is reached, we consider to following candidate for the feedback-Nash

$$\Lambda^* = \left(\Lambda_1^{s+1}, ..., \Lambda_N^{s+1}\right).$$

Doing so, we obtain a *true* (in the sense of convergence) Nash equilibrium for a *false* value function (until our value function also converges).

- ***Value iteration:*** once we have obtained a candidate for the Nash equilibrium (i.e., the previous game iteration loop ended), we update the value function at step *r+1* as following:

$$\left(\mathrm{V}_i^{r+1}\right)_j = \left\{hF(\Lambda^*)_j + (1-\rho_i h) I\left[\mathrm{V}_i^r\right](\Lambda^*)_j\right\}, \tag{27}$$

From a general point of view, our numerical scheme consists in a coupled system of the form $\mathrm{U}^{s+1} = G(\mathrm{U}^s; \mathrm{V}^r, \triangle)$,$\mathrm{V}^{r+1} = T(\mathrm{V}^r; \mathrm{U}^{s+1}, \triangle)$. It can be summarized as

$$\begin{cases} \text{While } error > \epsilon_1 \\ r = 0. \\ \text{For } s = 0, 1, ... \\ \Lambda^{s+1} = G(\Lambda^s; \mathrm{V}^r) \\ \text{If } \left\|\Lambda^{s+1} - \Lambda^s\right\| < \epsilon_2 \text{ Break For} \\ \text{End for} \\ \mathrm{V}^{r+1} = T\left(\mathrm{V}^r; \Lambda^{s+1}\right) \\ error = \left\|\mathrm{V}^{r+1} - \mathrm{V}^r\right\| \\ s = 0 \\ r = r + 1 \\ \text{End While} \end{cases} \tag{28}$$

Remark 1 *As done in Section 4 we compare our algorithm with a competitive one found in the literature (see [5]), we briefly recall the former algorithm main structure*

$$\begin{cases} \text{While } error > \epsilon_1 \\ r = 0 \\ \mathrm{V}^{r+1} = T\left(\mathrm{V}^r; \Lambda\right) \\ error = \left\|\mathrm{V}^{r+1} - \mathrm{V}^r\right\| \\ r = r + 1 \\ \text{End While} \end{cases} \tag{29}$$

The main difference between both methods, is the existence of the step called game iteration.

3.3 Algorithm Pseudo Code

Now, we present in detail a pseudo-code version of our algorithm presented in the previous Section.

- Initialize all parameters and counters:
  - Set: N the number of players; $0 < h < 1$ the time step; ρ_i the discount parameter; $\tau_i = 1 - \rho_i h$; $0 < \theta < 1$ the update parameter for the relaxation algorithm in *game iteration*. Fix $It_1max \in \mathbb{N}$ and $It_2max \in \mathbb{N}$, the maximum number of iterations for the two while loops defined below.
  - Initialize: $r = 0$ the *value iteration* step; $s = 0$ the *game iteration* step*;* the tolerance value for the stopping criteria: $tol \approx 0$ and $tolG \approx 0$, for Value Iteration and Game Iteration, respectively.
  - Define bounds for controls and state variables according to the experimental data.
- Build an initial approximation:
  - Generate a set $\mathcal{X}$ of Q random scattered points, with a uniform distribution, for each player from $\widetilde{X}$:

$$\mathcal{X} = ((\mathcal{X}_1)_1, ..., (\mathcal{X}_1)_Q, ..., (\mathcal{X}_N)_1, ..., (\mathcal{X}_1)_Q) \in \mathbb{R}^{Q \cdot N}$$

  - Generate a set of convenient initial values and controls:

$$\mathbf{V}^r = \left((\mathrm{V}_1)_1, ..., (\mathrm{V}_1)_Q, ..., (\mathrm{V}_N)_1, ..., (\mathrm{V}_N)_Q \right) \in \mathbb{R}^{Q \cdot N}.$$

    If there is no a priori information, they are set to zero.

$$\mathbf{\Lambda}^s = (\Lambda_1, ..., \Lambda_i, ..., \Lambda_N) \in \mathcal{M}_{Q \cdot N \times m}$$

- We define A, using radial basis functions (see (24))

$$(\mathrm{A})_{jl} = \Phi\left(\|\mathcal{X}_j - \mathcal{X}_l\| \right), \text{ for } j = 1, .., Q \text{ and } l = 1, ..., Q,$$

  - Determine $\bar{\lambda}_i^t$ (for $i = 1, ..., N$) by collocation (see (25)) satisfying:

$$\mathrm{A}\bar{\lambda}_i^r = \mathbf{V}_i^r$$

WHILE $(dif > tol)$ and $(r < It_1max)$
WHILE $(difG > tolG)$ and $(s < It_2max)$

- For $j = 1, ..., Q \cdot N$, perform:
  - Set $y = \mathcal{X}_j$
  - Set

$$(\mathbf{\Lambda}^s)_j = (\Lambda_1^s, ..., \Lambda_i^s, ..., \Lambda_N^s)_j$$

  - Apply the game iteration process, for each player $i = 1, ..., N$:
    - Obtain a state variable: $y^\sharp = y + hg(y, \left[\Lambda_i, \Lambda_{-i}^s\right]_j)$.
    - Get $\mathcal{A}_i = \Phi\left(\left\| \mathcal{X}_j - y^\sharp \right\| \right)$, $j = 1, ..., Q \cdot N$,
    - Compute$\left(\Lambda_i^{s+1}\right)_j = \theta\left(\Lambda_i^s\right)_j + (1-\theta)\arg\max_{\Lambda_i \in U_i} \left(h\mathrm{F}([\Lambda_i, \Lambda^s]_j) + (1 - \rho_i h)\left[\bar{\lambda_{\mathrm{i}}}^r \cdot \mathcal{A}_i\right]_j \right)$

* Set

$$\left(\mathbf{\Lambda}^{s+1}\right)_j = \left(\Lambda_1^{s+1}, ..., \Lambda_i^{s+1}, ..., \Lambda_N^{s+1}\right)_j$$

– Compute $difG = \left\|\Lambda^{s+1} - \Lambda^s\right\|$.

* Set $s = s + 1$

END WHILE

- Define$(\Lambda^*)_j = \left(\Lambda_1^{s+1}, ..., \Lambda_i^{s+1}, ..., \Lambda_N^{s+1}\right)_j$
- Set $y^\sharp = y + h \cdot g(y, (\Lambda^*)_j)$.
- Get $(\mathcal{A}_i)_j^* = \Phi\left(\left\|\mathcal{X}_j - y^\sharp\right\|\right),\ j = 1, ..., Q \cdot N$,
- Approximate value function : $I\left[\Lambda_i^*\right](\mathbf{V}_i^r)_j = \bar{\lambda}_i^r \cdot (\mathcal{A}_i^*)_j$
- Actualize the value function (Value Iteration process): $\left(\mathrm{V}_i^{t+1}\right)_j = \left\{h\mathrm{F}(\Lambda^*)_j + (1 - \rho_i h)(\bar{\lambda}_i^r \cdot \mathcal{A}_i^*)_j\right\}$
- Actualize $\bar{\lambda}_i^{r+1}$by collocation

$$\mathcal{A}_i \bar{\lambda}_i^{r+1} = \mathbf{V}_i^{r+1}$$

- Compute $dif = \left\|\mathrm{V}^{r+1} - \mathrm{V}^r\right\|$
- $r = r + 1$
- $s = 0$.

END WHILE

- In cases for which the algorithm converges, we return the equilibrium policies $\Lambda_1^*, ..., \Lambda_N^*$ and their associated optimal values: $\mathrm{V}_1^*, .., \mathrm{V}_N^*$, both depending on $\mathcal{X}$.
- By using RBF, we approximate the obtained feedback optimal controls to all values of the state variables which are in the bounded state space defined during the initialization step.

4 Numerical Experiments

In Sections 1 and 2, we have presented and developed the RaBVItG algorithm to solve differential games involving $N-$players pursuing a feedback-Nash. Now, in the current section, we apply the algorithm to several benchmark problems in order to evaluate its main properties.

Remark 2 *In a recent survey (see [13]), the existing numerical methods in the literature focus on some feasible subclass of games, regarding the ones we are dealing with in this work: two-person zero-sum games (see [9]) which can deal with the lack of differentiability of the value function by using the viscosity solution since this game can be reduced to a one dimensional game. Additionally, the other advances are in the line with linear quadratic models (see [8]), avoiding the interesting non linearity in the dynamic system or running cost. According to the authors, computational differential games is an area that needs to grow up.*

Here, we solve different test problems, selected from [5] and [8].

In order to check the efficiency of our approach, we compute: (i) the error of the method with respect to the analytic solution; and (ii) the CPU time and the number of iterations. Furthermore, we also compare our algorithm performances by comparing its results with to the one returned by the method proposed in [5], which is called in this work CCF (i.e., from the initial of the authors). To our knowledge, at this moment, it is the only algorithm found in the literature designed to solve similar problems. Both algorithms are based on the semi-lagrangian discretization of the problem and the design of value iterations as the main loop. However, they exhibit several relevant differences, such as:

- RaBVItG is a meshfree algorithm, and thus:
  1. only evaluates at some points of the state space,
  2. uses a RBF method to approximate functions,
  3. it does not require a discretization of the control space, so that we use a gradient algorithm to find the critical points at each algorithm step.
- RaBVItG incorporates an iteration inner loop, called game iteration, where players alternately update their strategies by averaging their current strategy with the best response to the other player's current strategy. The main advantage is that it finds the proper Nash equilibrium for an approximated value until convergence in the value iteration loop. Our hypothesis is that this step increases the accuracy and, thus, reduces the approximation error.

4.1 Numerical Test 1 (LQ, Scalar)

This example is introduced in [8]. In this case, we consider a two player ($N = 2$) scalar ($X \subseteq \mathbb{R}$) differential game, where $m = 2$. Let the following quadratic cost functional to be minimized, with $i = \{1, 2\}$:

$$J_i\left(y, u_1, u_2\right) = \int_0^\infty \left\{q_i x^2\left(t\right) + r_i u_i(t)^2\right\} \mathrm{d}t,$$

subject to the following linear dynamic system

$$\dot{x}\left(t\right) = ax\left(t\right) + b_1 u_1(t) + b_2 u_2\left(t\right),\ x\left(0\right) = y, y \in X.$$

According to [8], the analytic Feedback-Nash equilibria for this game is defined by the following linear expressions:

$$u_i^*(t) = f_i^* x^*(t) \equiv -\frac{b_i}{r_i} k_i x^*(t),$$

where k_1, k_2 are the equilibria for the following system of Ricatti ordinary differential equations, derived from the solution of the corresponding HJB equations

$$\begin{cases} \dot{k}_1 = -\frac{b_1^2}{r_1} k_1^2 - 2\frac{b_2^2}{r_2} k_1 k_2 + 2ak_1 + q_1, \\ \dot{k}_2 = -\frac{b_2^2}{r_2} k_2^2 - 2\frac{b_1^2}{r_1} k_1 k_2 + 2ak_2 + q_2. \end{cases} \tag{30}$$

Furthermore, an additional condition to be a stable equilibrium, is to satisfy the following inequality

$$a - \frac{b_1^2}{r_1} k_1 - \frac{b_2^2}{r_2} k_2 < 0, \tag{31}$$

where (31) divides the plane into "stable" and "unstable" regions. So, all the feedback Nash equilibrium are obtained as the intersection points of both hyperbolas in the "stable" region.

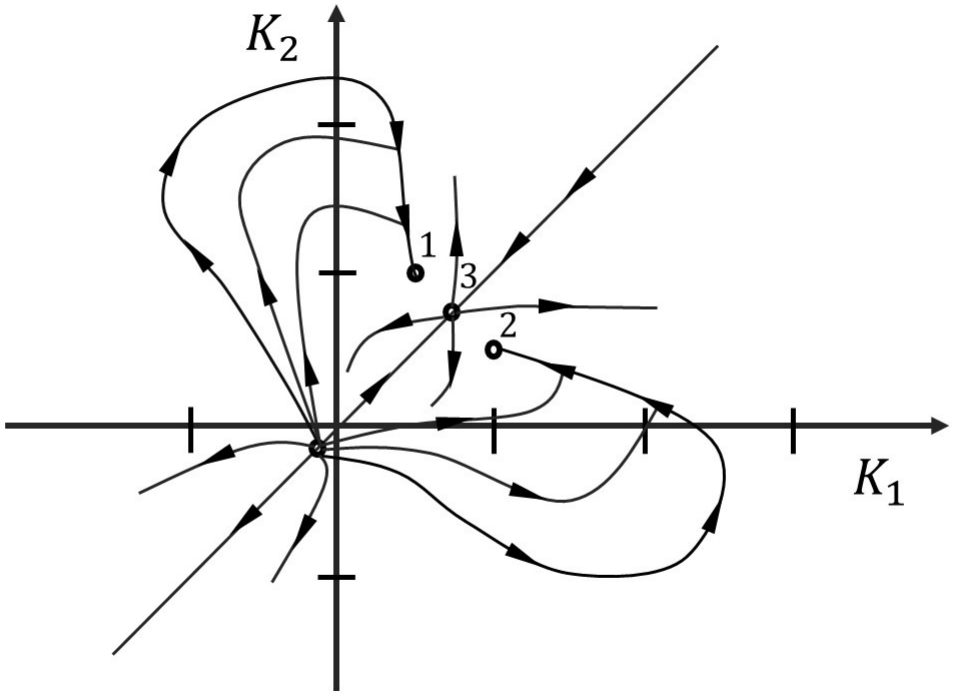

Figure 1: Numerical test 1: phase diagram of System (30).

Now, we consider the following particular parameter values: $a = 3, q_1 = q_2 = 1, b_1 = b_2 = 2, r_1 = r_2 = 1$ (see [8], p. 396). Equation (30) becomes

$$\dot{k}_1 = -4k_1^2 - 8k_1k_2 + 6k_1 + 1,$$
$$\dot{k}_2 = -4k_2^2 - 8k_1k_2 + 6k_2 + 1.$$

We obtain those equilibria values

$$\begin{pmatrix} k_1^{(1)} = 1/2 & k_2^{(1)} = 1 \\ k_1^{(2)} = 1 & k_2^{(2)} = 1/2 \\ k_1^{(3)} = 0.7287 & k_2^{(3)} = 0.7287 \\ k_1^{(4)} = -0.2287 & k_2^{(4)} = -0.2287 \end{pmatrix}.$$

However, according to Theorem 8.22 in [8], since we have here three feedback-Nash equilibria, then two of them are stable nodes and one is a saddle point. We note that the pair $(k_1^{(4)}, k_2^{(4)})$ does not satisfy the condition (31). In Figure (1) , we show the phase diagram with the three stable equilibria, where $k_i^{(1)}$ and $k_i^{(3)}$ are stable nodes and $k_i^{(2)}$ is a saddle point. Finally, $k_i^{(4)}$ is unstable.

Next, we analyze the performance of our algorithm to solve this problem and compare it with CCF, mentioned previously. To do so, we choose three different starting points which are in the basin of attraction of the different stable feedback-Nash equlibria. As it can be seen on Figure 1, equilibria $k^{(2)}$ have a symmetric behavior with respect to $k^{(1)}$ and, so, the results will remain the same. Thus, we only analyze two of the three equilibria. In order to analyze the performance of both algorithms,

		$A = (-2x, -x)$			$B = (-1.4574x, -1.4574x)$		
Time Step	Algorithm	$(-2.75x, -0.5x)$	$(-2x, -x)$	$(-4x, -0.25x)$	$(-x, -x)$	$(-1.4574x, -1.4574x)$	$(-4x, -4x)$
$h = 0.1$	RaBVitG	0.1427 14	0.3212 10	0.2227 26	0.1039 10	0.1025 10	0.0927 5
	CCF	0.2832 22	0.2832 10	0.2331 24	0.1161 9	0.1161 6	0.1161 12
$h = 0.05$	RaBVitG	0.0809 14	0.0159 4	0.0793 74	0.0109 19	0.0104 19	0.0089 7
	CCF	0.1955 67	0.2206 22	0.1704 52	0.0535 20	0.0535 8	0.0535 12
$h = 0.01$	RaBVitG	0.0267 847	0.0139 136	0.0267 912	0.009 57	0.009 56	0.0068 23
	CCF	0.0451 440	0.0450 68	0.0840 584	0.0284 56	0.0284 25	0.0284 19

Table 1: Numerical Test 1 LQ, Scalar: results obtained with RaBVItG and CCF. In each cell, we report two values: (i) the upper number represents $L^{\infty}_{error\ A}$ or $L^{\infty}_{error\ B}$ depending on the case; (ii) and the bottom number represents the number of iterations the algorithm require for convergence.

we run them three times starting from the following initial points: the corresponding equilibrium and points in the attraction domain of each equilibria (see Figure 1).

We define our state variable as $0 \leq x \leq 1$ and we discretize it by using $Q = 9$ points, so $\mathcal{X} = [0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1]$. Regarding the control space, it is bounded by considering values in $[-5, 5]$. RaBVItG does not need any grid in the control space. In the case of CCF, we have generated a uniform grid of 20 elements from the defined space.

In Table 1, we summarize the results obtained with both algorithms (i.e., RaBVItG and CCF) and their performances to reach the two equilibria of interest. For the three different starting equilibrium for each player, and, for each starting point, we report the errors values

$$L^{\infty}_{error\ A} = \max\left[-2x - (u^1_A), -x - (u^2_A)\right]$$

where u^i_A is the vector of solutions (for $i = 1, 2$) obtained by the algorithms trying to achieve the equilibrium $A = (-2x, -x)$, and

$$L^{\infty}_{error\ B} = \max\left[-1.4574x - (u^1_B), -1.4574x - (u^2_B)\right],$$

where u^i_B are the numerical solutions obtained for the equilibrium $B = (-1.4574x, -1574x)$, starting from the different initial values.

As it can be seen on Table 1, both algorithms show a similar qualitative pattern: the smaller is the value of h, the smaller is the error and the higher is the number of iterations needed. It is interesting to remark that our algorithm, RaBVitG obtains lower errors but requires, in general, more iterations than CCF. However, as we point out in the next experiments, the computational time of RaBVitG is dramatically lower than the one of the CCF algorithm. This tends to show that RaBVitG seems to be more efficient and accurate than CCF.

On the other hand, Figure 2 shows how both algorithms converge to different equilibrium, depending on the starting point. To perform this experiment, we have run both algorithms using a mesh of 16^2 initial conditions from the set $\{-4, -3.5, ..., -0.5\} \times \{-4, -3.5, ..., -0.5\}$. As we can see on Figure 1, for each plot, the white region represents the initial points that converge to the corresponding equilibrium (i.e, they satisfy the tolerance criterium of convergence with respect to the point represented by a circle). Additionally, the grey region is composed by the rest of initial points that does not satisfy this tolerance criterium. Thus, the white region can be interpreted as a numerical approximation of the basin of attraction of each equilibrium.

Comparing both algorithms, RaBVItG has less problems than CCF to identify the correct equilibrium in the area $(f_1, f_2) \in [0, -1] \times [0, -1]$. Focusing on the saddle equilibrium, the stable manifold is situated on the bisector, so any trajectory starting out of this bisector should converge to the closest stable node (i.e, (-2,-1) or (-1,-2)). As it can be seen on Figure 2, CCF exhibits more wrong equilibria than RaBVItG. When starting from the attraction region of a stable equilibrium, for some initial points

both algorithms converge to the saddle point. However, RaBVItG performs better than CCF, since the convergence band around this equilibrium is finer.

4.2 Numerical Test 2 (non LQ, Scalar)

The next experiment was proposed in [5]. Again, we present a two-player scalar differential game, with the following cost functional to minimize, with $i = \{1, 2\}$

$$J_i\left(y, u_1, u_2\right) = \int_0^\infty \left\{ s_i x\left(t\right) + \frac{u_i(t)^2}{2} \right\} \mathrm{d}t,$$

subject to the dynamic system

$$\dot{x}\left(t\right) = u_1(t) + u_2\left(t\right),\ x\left(0\right) = y.$$

with $y \in \mathbb{R}$.

According to [5], we obtain the analytical value functions per player attained to their feedback-Nash equilibrium,

$$v_1^*(x) = s_1 x - s_1 s_2 - \frac{s_1^2}{2}$$

and

$$v_2^*(x) = s_2 x - s_1 s_2 - \frac{s_2^2}{2}$$

Here, we consider $s_1 = -1, s_2 = 2$. It is straight forward to deduce, once we know the true value function and using the corresponding HJB equations, the optimal feedback-Nash policies remain constant for all $x(t)$ values

$$u_1^* = 1, u_2^* = -2. \tag{32}$$

Focusing on the RaBVItG and CCF configurations, we use the following specifications: the admissible values for the state variable is in $[-50, 50]$. We discretize it using $Q = 8$ equispaced points: $x = \{-50, -37.5, -25, -12.5, 0, 12.5, 25, 37.5, 50\}$. The set of admissible controls is in $[-5, 5]$. Additionally, only in the case of CCF, we discretize the control variable using a grid of M equispaced points. We choose $M = \{50, 150, 400\}$ in order to compare the effect of improving the control space discretization. We use an interpolator based on splines to approximate the value function in the corresponding algorithm step.

In Figure (3), we depict the different errors in infinity-norm between the numerical feedback-Nash policies and the real ones (32). As we can see on this figure, the greater M is, the better approximation to the analytical solution is. Indeed, in the case of $M = 400$, CCF produces slightly better results than RaBVItG. However, there is an important trade off between improving the error measures and the number of points that CCF needs. Finally, reported on Table (2), our algorithm is clearly more time efficient, and require, approximately, the same quantity of iterations in value than CCF.

4.3 Numerical Test 3: (LQ, Scalar, Three Players)

We consider the following three-players minimization differential game, for $i = \{1, 2, 3\}$

$$J_i\left(y, u_1, u_2, u_3\right) = \int_0^\infty \left\{ q_i x^2\left(t\right) + r_i u_i(t)^2 \right\} \mathrm{d}t,$$

subject to the following dynamic system

$$\dot{x}\left(t\right) = a x\left(t\right) + b_1 u_1(t) + b_2 u_2\left(t\right) + b_3 u_3(t),\ x\left(0\right) = y.$$

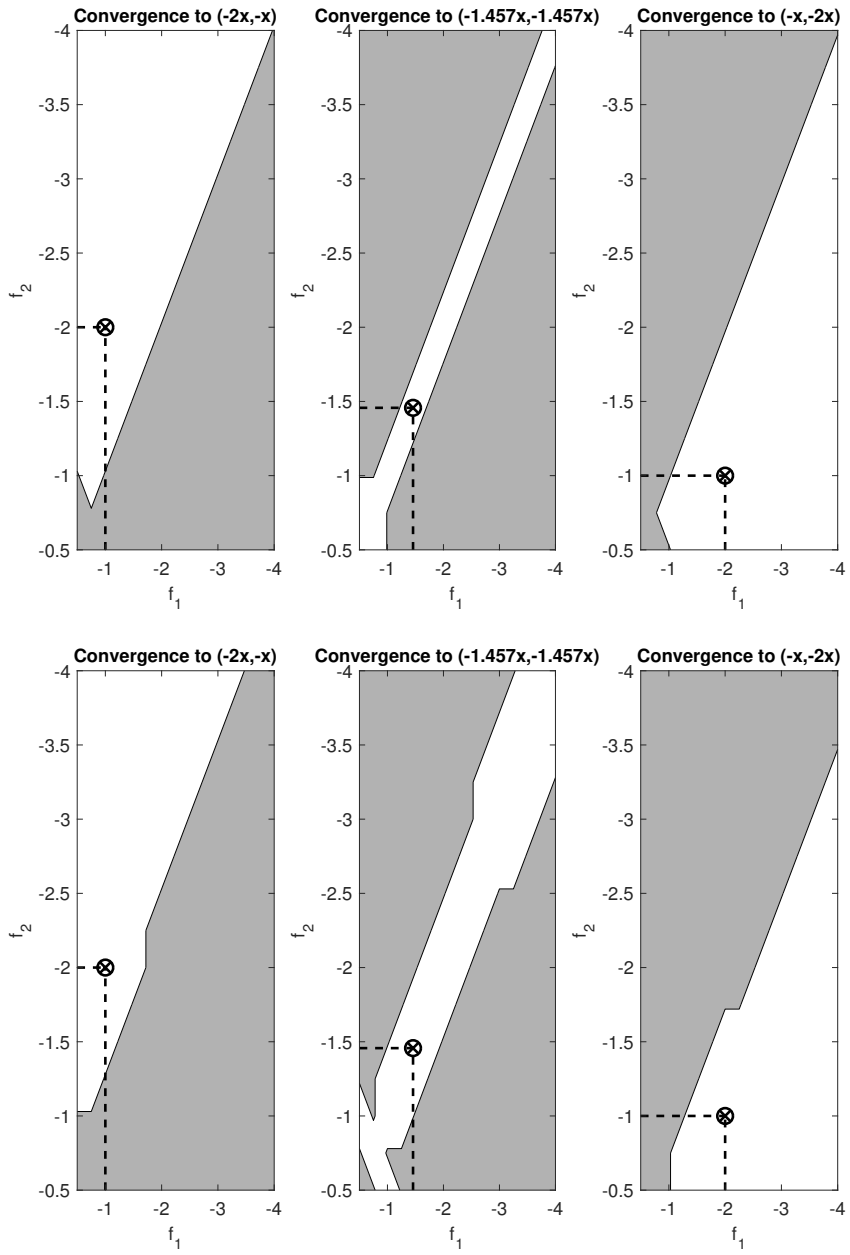

Figure 2: Numerical Test 1 (LQ, Scalar): Convergence of the algorithm towards Nash equilibria: (top) RaBVItG and (bottom) CCF). f_1, f_2 represent the starting points $u_i^0(t) = f_i^0 x(t)$, $i = 1, 2$. The white region means that the algorithm starting from a point inside this area tends to the considered equilibrium (represented by a circle). The grey region means that the algorithm starting from this area tends to another equilibrium.

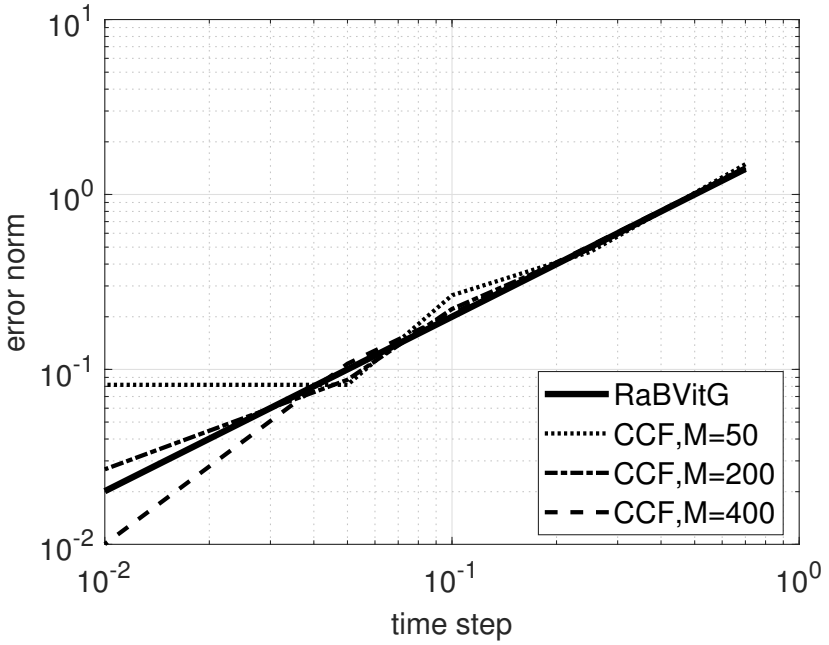

Figure 3: Numerical Test 2 (non LQ, scalar):
L^∞ errors between numerical and analytical feedback-Nash policies obtained considering RaBVItG and CCF with $M = \{50, 150, 400\}$.

		$h = 0.7$	$h = 0.25$	$h = 0.1$	$h = 0.05$	$h = 0.01$
RaBVitG	-	$\underset{3}{0.048}$	$\underset{8}{0.052}$	$\underset{27}{0.054}$	$\underset{68}{0.062}$	$\underset{501}{0.075}$
CCF	$M = 50$	$\underset{3}{1.049}$	$\underset{8}{6.036}$	$\underset{27}{24.408}$	$\underset{69}{65.3135}$	$\underset{507}{523.53}$
	$M = 150$	$\underset{3}{3.30}$	$\underset{8}{14.87}$	$\underset{27}{61.03}$	$\underset{68}{200.8}$	$\underset{503}{1702.1}$
	$M = 400$	$\underset{3}{3.37}$	$\underset{8}{19.77}$	$\underset{27}{81.2}$	$\underset{68}{214.9}$	$\underset{502}{1763.3}$

Table 2: Numerical Test 2 (non LQ, scalar): performance comparison between RaBVitG and CCF. The upper number report the CPU time (s) and the bottom number the number of iterations.

		$h = 0.7$	$h = 0.25$	$h = 0.1$	$h = 0.05$	$h = 0.01$
RaBVitG	-	0.0049 8	0.0050 17	0.0058 36	0.0048 55	0.0081 215
CCF	$M = 50$	0.8453 4	1.0181 5	3.2296 14	9.003 36	177.2065 668
	$M = 150$	1.5780 4	2.2935 5	8.4177 13	22.1266 31	180.2918 238
	$M = 400$	4.30 4	6.65 5	22.72 13	61.06 31	477.56 234

Table 3: Numerical Test 3 (LQ, Scalar, Three Players): performance comparison between RaBVitG and CCF. The upper number report the CPU time (s) and the bottom number the number of iterations.

with $y \in \mathbb{R}$. Additionally, the considered parameter values are $a = -1, b_1 = 1, b_2 = 1, b_3 = 0.5, q_1 = 2, r_1 = 1, q_2 = 2, r_2 = 2, q_3 = 1, r_3 = 1$.

According to [8], this model has only one feedback Nash-equilibrium, which consists in

$$u_1^*(t) = 0.6202x(t)\ u_2^*(t) = -0.2805x(t)\ u_3^*(t) = 0.0436x(t)$$

We set our admissible values for the state variable in $[0, 1]$ and we discretize it by using $Q = 9$ points, so $\mathcal{X} = [0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1]$. Regarding the control space, we consider values in $[-2, 2]$. For the CCF algorithm , again we discretize the control variable using a grid of M equally spaced points. As previously, we choose $M = \{50, 150, 400\}$, and we use an interpolator based on splines.

As we can see on Table 3, RaBVItG again exhibits better computational times, showing the greatest efficiency for all the different time discretization values. Furthermore, in Figure 4, we observe that even with $M = 400$, RaBVItG produces errors smaller than using CCF. The difference with respect to Test 2 is that the solution, in this case, is not a constant one, being a linear function.

4.4 Numerical Test 4: (Non Scalar, Two Players)

This last minimization Differential Game is interesting as is not a scalar one, because its state variable has two dimensions. The running-cost for $i = \{1, 2\}$ is

$$J_i\left(y, u_1, u_2\right) = \begin{cases} \int_0^\infty e^{-t}\left\{\sqrt{x_1^2\left(t\right) + x_2(t)^2}\right\} \mathrm{d}t & if\ \sqrt{x_1^2\left(t\right) + x_2(t)^2} > 1 \\ 0 & otherwise \end{cases}$$

subject to the dynamic system

$$\begin{cases} \dot{x}_1\left(t\right) = u_2\left(t\right) \\ \dot{x}_2\left(t\right) = u_1\left(t\right) \end{cases} \quad x\left(0\right) = y.$$

This test case was introduced in [5] but, unfortunately, the authors do not provide any analytical solution to this differential game in order to compare with. However, they show a figure with a value function comparable to the ones reported on Figure 5. We run CCF and RaBVItG using different mesh sizes (in the case of CCF, $M_1 = 20^2, 30^2, 40^2, 50^2$) and meshfree points ($M_2 = 20, 30, 40, 50$) in the case of RaBVItG. In Figure 5 we represent the different value functions for Player 1 (Player 2 has the same result since it is a symmetric game). The fact that CCF requires a mesh for the state variables (and controls) affects clearly to the CPU time required to achieve convergence. Indeed, in order to compare the solution of RaBVItG with CCF algorithms, once we obtain the value function per player (as a M_2- dimensional array) we interpolate our solution into the mesh defined in CCF using, again, RBF methods. Both algorithms converge to the same solution as the mesh size increases. However, as shown on Table 4, CCF method needs, in general, less value iterations but is much computationally expensive than RaBVItG costly (around 10 times greater).

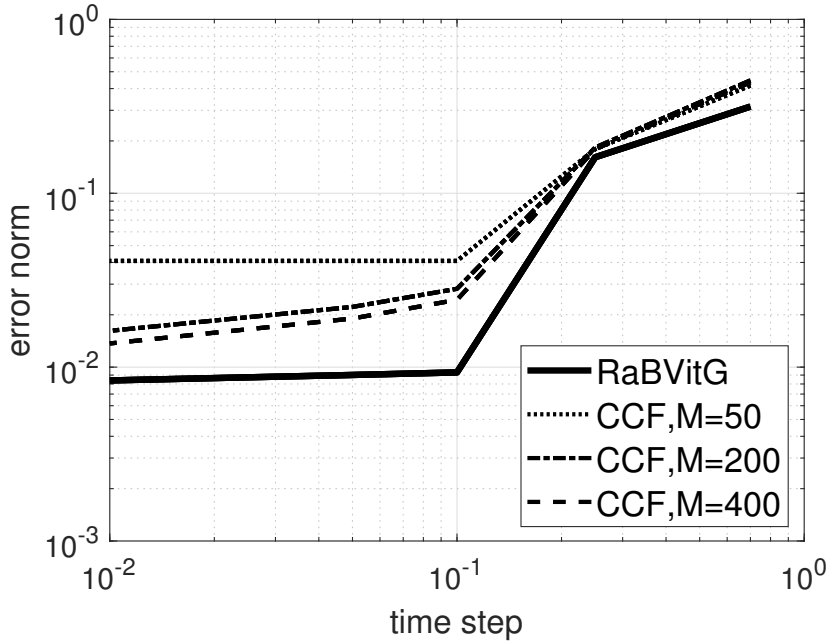

Figure 4: Numerical Test 3 (LQ, scalar, Three players):
L^∞ errors between numerical and analytical Feedback-Nash policies obtained considering RaBVItG and CCF with $M = \{50, 150, 400\}$.

	CCF				RaBVitG			
size	20^2	30^2	40^2	50^2	20	30	40	50
CPU time	2140	5140.3	9623.6	12854	479.11	603.08	734.8	1272.8
Its	333	287	266	248	542	457	397	275

Table 4: Numerical Test 4 (non Scalar, Two Players): performance comparison between RaBVitG and CCF: mesh size M, CPU time (s) and number of iterations (Its).

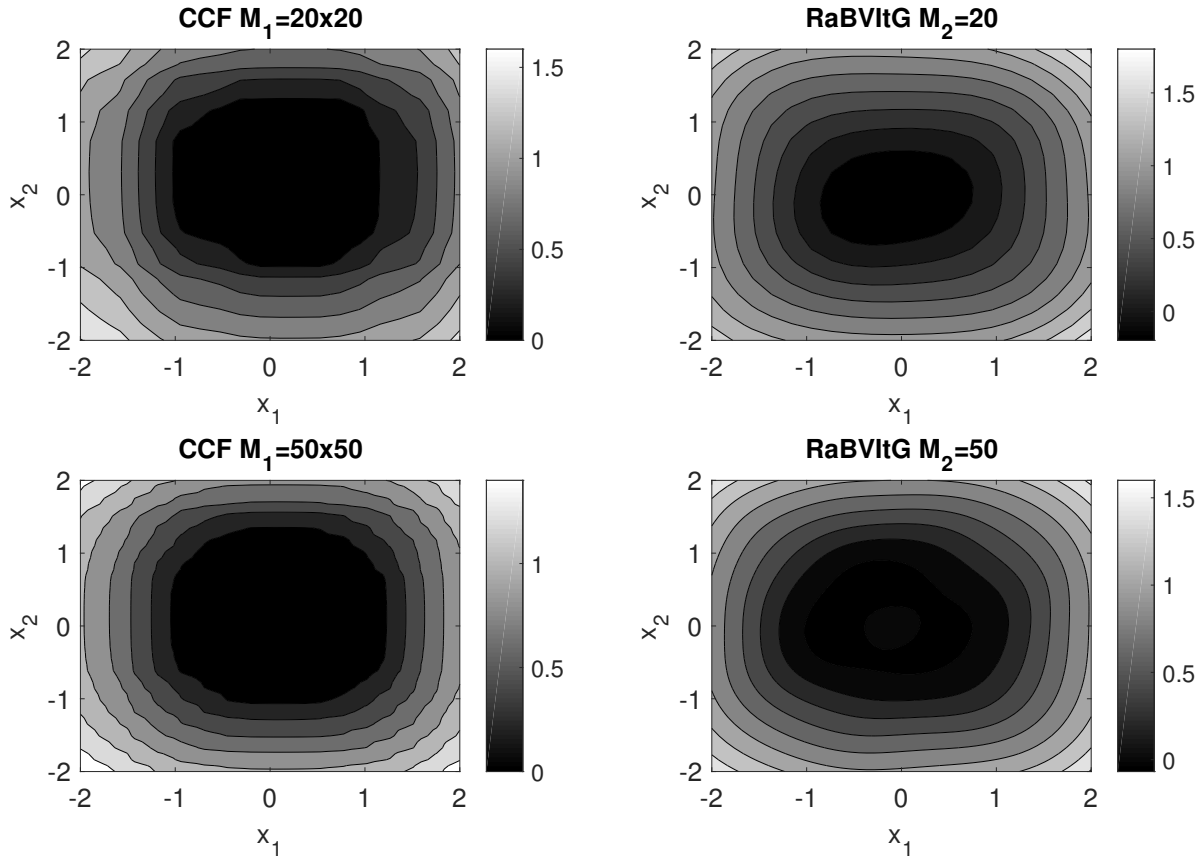

Figure 5: Numerical Test 4 (non Scalar, Two Players): value functions for player 1 returned by RaBVItG and CCF.

5 Conclusions

In this work, we have developed a novel numerical algorithm (RaBVItG) for solving multi-players feedback Nash differential games. RaBVItG is an algorithm based on radial basis function approximators (known to work properly without mesh), value iteration (a classical approach in dynamic programming), reinforcement learning (to solve a class of Hamilton-Jacobi-Bellman equations) and game iteration (to obtain at each value iteration step, the corresponding feedback Nash equilibrium).

The general purpose of this algorithm is to deal with multi-players problems, with two or more players and allowing the dimension of the control space for each player to be greater than one. For this reason, we have designed an algorithm which is mesh free for both state and control spaces. Additionally, we have also selected the semi-lagrangian discretization for the HJB equation due to its fine results generally reported in the literature for problems similar to the differential games studied in this paper.

We have validated RaBVItG by comparing the obtained results with the ones returned by another algorithm published in the literature (here, called CCF, see [5]). This particular algorithm is based on meshes in the state and control spaces and the way to get the Nash solution is relatively different from our approach. Indeed, CCF performs value iteration steps for a particular player fixing at every step the controls for the remaining players, until convergence. In our case, RaBVItG alternates value iteration steps with a new step called game iteration. During this step, fixing the value for each player, the algorithm iterates until convergence to a Nash equilibrium. So, we find the true (in the sense of convergence) Nash equilibrium for an approximated value (again, in the sense of convergence). As shown in our numerical experiments, our approach seems to exhibit a better accuracy when approximating the true equilibrium. Indeed, the different experiments performed in order to compare both algorithms tend to show that RaBVItG is, on average, 10 times faster than CCF and, in some of the experiments, with a smaller quantity of data points, RaBVitG obtains smaller errors.

Additionally, RaBVItG can be implemented in real problems with more than two players and more than two controls per player, which are problems difficult to solve with CCF.

Future lines of work on RaBVItG include the study of a stochastic version (e.g., including a multivariate diffusion term in the state equations) and different game iterations schemes in order to find other typical equilibria studied in the literature (e.g., Stackelberg, or Pareto).

Acknowledgment

This work was carried out thanks to the financial support of the Spanish "Ministry of Economy and Competitiveness" under project MTM2015-64865-P; the research group MOMAT (Ref. 910480) supported by "Banco Santander" and "Universidad Complutense de Madrid"; and the "Junta de Andalucía" and the European Regional Development Fund through project P12-TIC301.

References

[1] Martino Bardi and Italo Capuzzo-Dolcetta. *Optimal control and viscosity solutions of Hamilton-Jacobi-Bellman equations*. Springer Science & Business Media, 2008.

[2] Dimitri P. Bertsekas. *Dynamic programming and optimal control*. Number 3 in 1. Athena scientific Belmont, MA, 2005.

[3] Alberto Bressan. Noncooperative differential games. *Milan Journal of Mathematics*, 79(2):357–427, 2011.

[4] Alberto Bressan and Wen Shen. Small bv solutions of hyperbolic noncooperative differential games. *SIAM journal on control and optimization*, 43(1):194–215, 2004.

[5] Simone Cacace, Emiliano Cristiani, and Maurizio Falcone. Numerical approximation of nash equilibria for a class of non-cooperative differential games. *arXiv preprint arXiv:1109.3569*, 2011.

[6] Michael G. Crandall and Pierre-Louis Lions. Viscosity solutions of hamilton-jacobi equations. *Transactions of the American mathematical society*, 277(1):1–42, 1983.

[7] Engelbert J. Dockner, Steffen Jorgensen, Ngo Van Long, and Gerhard Sorger. *Differential games in economics and management science*. Cambridge University Press, 2000.

[8] Jacob Engwerda. *LQ dynamic optimization and differential games*. John Wiley & Sons, 2005.

[9] Maurizio Falcone. Numerical methods for differential games based on partial differential equations. *International Game Theory Review*, 8(02):231–272, 2006.

[10] Maurizio Falcone and Roberto Ferretti. *Semi-Lagrangian approximation schemes for linear and Hamilton-Jacobi equations*, volume 133. SIAM, 2013.

[11] Gregory E. Fasshauer. *Meshfree approximation methods with MATLAB*, volume 6. World Scientific, 2007.

[12] Lars Grüne and Willi Semmler. Using dynamic programming with adaptive grid scheme for optimal control problems in economics. *Journal of Economic Dynamics and Control*, 28(12):2427–2456, 2004.

[13] Steffen Jørgensen and Georges Zaccour. Developments in differential game theory and numerical methods: economic and management applications. *Computational Management Science*, 4(2):159–181, 2007.

[14] Jacek B Krawczyk and Stanislav Uryasev. Relaxation algorithms to find nash equilibria with economic applications. *Environmental Modeling & Assessment*, 5(1):63–73, 2000.

[15] Warren B. Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703. John Wiley & Sons, 2007.

[16] Leigh Tesfatsion and Kenneth Juddn (Eds.). *Handbook of computational economics (Vol. 2)*. Elsevier, 2006.