Johann Radon Institute for
Computational and Applied Mathematics
Austrian Academy of Sciences (ÖAW)

An efficient policy iteration algorithm for dynamic programming equations

Alessandro Alla, Maurizio Falcone, Dante Kalise

RICAM-Report 2014-15

AN EFFICIENT POLICY ITERATION ALGORITHM FOR DYNAMIC PROGRAMMING EQUATIONS *

ALESSANDRO ALLA† MAURIZIO FALCONE‡ DANTE KALISE§

Abstract. We present an accelerated algorithm for the solution of static Hamilton-Jacobi-Bellman equations related to optimal control problems. Our scheme is based on a classic policy iteration procedure, which is known to have superlinear convergence in many relevant cases provided the initial guess is sufficiently close to the solution. This limitation often degenerates into a behavior similar to a value iteration method, with an increased computation time. The new scheme circumvents this problem by combining the advantages of both algorithms with an efficient coupling. The method starts with a coarse-mesh value iteration phase and then switches to a fine-mesh policy iteration procedure when a certain error threshold is reached. A delicate point is to determine this threshold in order to avoid cumbersome computations with the value iteration and at the same time, to ensure the convergence of the policy iteration method to the optimal solution. We analyze the methods and efficient coupling in a number of examples in different dimensions, illustrating their properties.

Key words. policy iteration, dynamic programming, semi-Lagrangian schemes, Hamilton-Jacobi equations, optimal control

AMS subject classifications. 65N55, 49L20

1. Introduction. The numerical solution of optimal control problems is a crucial issue for many industrial applications such as aerospace engineering, chemical processing, power systems, and resource economics, among many others. In some cases the original problem comes from a different setting, e.g. when one has to fit a given set of data or has to solve a shape optimization problem, but has been reformulated in terms of a control problem for an appropriate dynamic and cost functional. The typical goal is then to compute an optimal trajectory for the controlled system and its corresponding optimal control. In the framework of open-loop controls the classical solution is based on the Pontryagin Maximum Principle which leads to the solution of a two-point boundary value problem for the coupled state/costate system. The numerical solution can be obtained via a shooting method. Despite its simplicity and mathematical elegance, this approach is not always satisfactory because the initialization of the shooting method can be a difficult task, mainly for the costate variables, besides having the usual limitations of open-loop controls. It is well known that the Dynamic Programming (DP) approach introduced by Bellman [5] produces optimal controls in feedback form, looking more appealing in terms of online implementations and robustness. However, the synthesis of feedback controls require the previous knowledge of the value function and this is the major bottleneck

†DEPARTMENT OF MATHEMATICS, UNIVERSITÄT HAMBURG, BUNDESSTR. 55, 20146 HAMBURG, GERMANY. ALESSANDRO.ALLA@UNI-HAMBURG.DE

‡(CORRESPONDING AUTHOR) DIPARTIMENTO DI MATEMATICA, SAPIENZA - UNIVERSITÀ DI ROMA, P.LE ALDO MORO 2, 00185 ROME, ITALY. FALCONE@MAT.UNIROMA1.IT

§JOHANN RADON INSTITUTE FOR COMPUTATIONAL AND APPLIED MATHEMATICS, ALTENBERGERSTRAßE 69, 4040 LINZ, AUSTRIA. DANTE.KALISE@OEAW.AC.AT

*The authors wish to acknowledge the support of the following grants: AFOSR Grant no. FA9550-10-1-0029, ITN - Marie Curie Grant n. 264735-SADCO , START-Project Y432-N15 "Sparse Approximation and Optimization in High-Dimensions" and SAPIENZA 2009 "Analisi ed approssimazione di modelli differenziali nonlineari in fluidodinamica e scienza dei materiali". The authors also wish to thank the CASPUR Consortium for its technical support.

for the application of DP. In fact, the value function of an optimal control problem is known to be only Lipschitz continuous even when the data is regular. The characterization of the value function is obtained in terms of a first order nonlinear Hamilton-Jacobi-Bellman (HJB) partial differential equation. In the last twenty years, the DP approach has been pursued for all the classical control problems in the framework of viscosity solution introduced by Crandall and Lions in the 80's (see [4] for a comprehensive illustration of this approach). Moreover, several approximation schemes have been proposed for this class of equations, ranging from finite differences to semi-Lagrangian and finite volume methods. Some of these algorithms converge to the value function but their convergence is slow. The so-called *curse of the dimensionality*, namely the fact that the dimension of the partial differential equation characterizing the value function increases as the dimension of the state space does, constitutes a major computational challenge towards a practical implementation of numerical algorithms for optimal control design based on viscosity solutions of HJB equations.

Our main contribution in this paper is a new accelerated algorithm which can produce an accurate approximation of the value function in a reduced amount of time in comparison to the currently available methods. Furthermore, the proposed scheme can be used in a wide variety of problems connected to static HJB equations, such as infinite horizon optimal control, minimum time control and some cases of pursuit-evasion games. The new method couples two ideas already existing in the literature: the value iteration method (VI) and the policy iteration method (PI) for the solution of Bellman equations. The first is known to be slow but convergent for any initial guess, while the second is known to be fast when it converges (but if not initialized correctly, convergence might be as slow as the value iteration). The approach that we consider relates to multigrid methods (we refer to Santos [27] for a brief introduction to subject in this context), as the coupling that we introduce features an unidirectional, two-level mesh. The work by Chow and Tsitsiklis [13] exploits a similar idea with a value iteration algorithm. However, as far as we know the efficient coupling between the two methods has not been investigated.

To set this paper into perspective, we must recall that algorithms based on the iteration in the space of controls (or policies) for the solution of HJB equations has a rather long history, starting more or less at the same time of dynamic programming. The PI method, also known as Howard's algorithm [21], has been investigated by Kalaba [22] and Pollatschek and Avi-Itzhak [25] who proved that it corresponds to the Newton method applied to the functional equation of dynamic programming. Later, Puterman and Brumelle [26] have given sufficient conditions for the rate of convergence to be either superlinear or quadratic. More recent contributions on the policy iteration method and some extensions to games can be found in Santos and Rust [29] and Bokanowski et al. [7]. Results on its numerical implementation and diverse hybrid algorithms related to the proposed scheme have been reported in Capuzzo-Dolcetta and Falcone [11], González and Sagastizábal [20], Grüne [19] and in the recent monograph by Falcone and Ferretti [15].

Finally, we should mention that an acceleration method based on the the set of subsolutions has been studied in Falcone [14] (see also Tidball and González [32] for a specific application to the Isaacs equation). More in general, dealing with acceleration methods for Hamilton-Jacobi-Bellman equations, we should also mention approaches based on domain decomposition algorithms as in Falcone et al. [16] and more recently by Cacace et al. [9], on geometric considerations as in Botkin, et al. [8], and

those focusing on the localization of numerical schemes which leads to Fast Marching Methods. This approach has shown to be very effective for level-set equations related to front propagation problems (see e.g. the book by Sethian [30]), i.e. eikonal type equations. At every iteration, the scheme is applied only on a subset of nodes (localization) which are the nodes close to the front, the so-called *narrow band*. The remaining part of the grid is divided into two parts: the accepted region, where the solution has been already computed, and the far region where the solution will be computed little by little in the following iterations. At every iteration, one node is accepted and moved from the narrow band to the accepted region; the narrow band is then updated adding the first neighbors of that node (which before where in the far region). For eikonal-type of equations these methods converge in finite number of iterations to the correct viscosity solution and have a very low complexity (typically $O(N \ln(N))$, where N is the cardinality of the grid). More recently, several efforts have been made to extend these methods to more complex problems where the front propagation is anisotropic [31] and/or to more general Hamilton-Jacobi equations as in [3]. However, their implementation is rather delicate and their convergence to the correct viscosity solution for general Hamilton-Jacobi equations is still an open problem; we refer to [10] for a an extensive discussion and several examples of these limitations.
The paper is organized as follows. In Section 2, we introduce some basic notions for optimal control synthesis by the dynamic programming principle. Section 3 contains the core of the proposed accelerated method and discuss practical implementation details. Finally, Section 4 shows our numerical results on a number of different examples concerning infinite horizon optimal control, minimum time control, and some further extensions towards the optimal control of partial differential equations. In these series of tests we discuss several properties of the proposed scheme and perform comparisons with the different techniques presented in the article.

2. Dynamic programming in optimal control and the basic solution algorithms. In this section we will summarize the basic results for the two methods as they will constitute the building blocks for our new algorithm. The essential features will be briefly sketched, and more details can be found in the classical books by Bellman [5], Howard [21] and for a more recent setting in the framework of viscosity solutions, in [11] and [4].
Let us first present the method for the classical *infinite horizon problem*. Let the dynamics be given by

$$\begin{cases} \dot{y}(t) = f(y(t), \alpha(t)) \\ y(0) = x \end{cases} \tag{2.1}$$

where $y \in \mathbb{R}^d$, $\alpha \in \mathbb{R}^m$ and $\alpha \in \mathcal{A} \equiv \{a : \mathbb{R}_+ \to A, \text{measurable}\}$. If f is Lipschitz continuous with respect to the state variable and continuous with respect to (y, α), the classical assumptions for the existence and uniqueness result for the Cauchy problem (2.1) are satisfied. To be more precise, the Carathéodory theorem (see [17] or [4]) implies that for any given control $\alpha(\cdot) \in \mathcal{A}$ there exists a unique trajectory $y(\cdot; \alpha)$ satisfying (2.1) almost everywhere.
Let us introduce the *cost functional* $J : \mathcal{A} \to \mathbb{R}$ which will be used to select the "optimal trajectory". For infinite horizon problem the functional is

$$J_x(\alpha(\cdot)) = \int_0^\infty g(y(s), \alpha(s)) e^{-\lambda s} ds \,, \tag{2.2}$$

where g is Lipschitz continuous in both arguments and $\lambda > 0$ is a given parameter. The function g represents the running cost and λ is the discount factor which allows to compare the costs at different times by rescaling the costs at time 0. From the technical point of view, the presence of the discount factor guarantees that the integral is finite whenever g is bounded, i.e. $||g||_\infty \leq M_g$. Let us define the value function of the problem as

$$v(x) = \inf_{\alpha(\cdot)\in\mathcal{A}} J_x(\alpha(\cdot))\,. \tag{2.3}$$

It is well known that passing to the limit in the Dynamic Programming Principle one can obtain a characterization of the value function in terms of the following first-order nonlinear Bellman equation

$$\lambda v(x) + \max_{a\in A}\{-f(x,a)\cdot Dv(x) - g(x,a)\} = 0, \qquad \text{for } x\in\mathbb{R}^d\,. \tag{2.4}$$

Several approximation schemes on a fixed grid G have been proposed for (2.4). Here we will use a semi-Lagrangian approximation based on a Discrete Time Dynamic Programming Principle. This leads to

$$v_{\Delta t}(x) = \min_{a\in A}\{(1-\lambda\Delta t)v_{\Delta t}\left(x+\Delta t f\left(x,a\right)\right) + \Delta t g\left(x,a\right)\}\,, \tag{2.5}$$

where $v_{\Delta t}(x)$ converges to $v(x)$ when $\Delta t \to 0$. A natural way to solve (2.5) is to write it in fixed point form

$$V_i = \min_{a\in A}\{(1-\lambda\Delta t)I[V]\left(x_i+\Delta t f\left(x_i,a\right)\right) + \Delta t g\left(x_i,a\right)\}\,, \quad i=1,\dots,N_G \tag{2.6}$$

where $\{x_i\}_{i=1}^{N_G}$ are the grid nodes, V_i is the approximate value for $v(x_i)$ and $I[V]:\mathbb{R}^d\to\mathbb{R}$ represents an interpolation operator defining, for every point x, the polynomial reconstruction based on the values V_i (see [4, Appendix A] for more details). Finally, one obtains the following algorithm:

Algorithm 1: Value Iteration for infinite horizon optimal control **(VI)**

Data: Mesh G, Δt, initial guess V^0, tolerance ϵ.
while $||V^{k+1}-V^k|| \geq \epsilon$ **do**
  forall the $x_i \in G$ **do**
    Solve: $V_i^{k+1} = \min_{a\in A}\{(1-\lambda\Delta t)I\left[V^k\right]\left(x_i+\Delta t f\left(x_i,a\right)\right) + \Delta t g\left(x_i,a\right)\}$ (2.7)
  end
  $k = k+1$
end

Here V_i^k represents the values at a node x_i of the grid at the k-th iteration; without loss of generality, throughout this paper we will assume that the numerical grid G is a regular equidistant array of points with mesh spacing denoted by Δx, and we consider a multilinear interpolation operator. Extensions to nonuniform grids and high-order interpolants can be performed in a straightforward manner.
Algorithm 1 is referred in the literature as the *value iteration method* because, starting from an initial guess V^0, it modifies the values on the grid according to the nonlinear rule (2.7). It is well-known that the convergence of the value iteration can be very slow, since the contraction constant $1-\lambda\Delta t$ is close to 1 when Δt is close to 0. This

means that a higher accuracy will also require more iterations. Then, there is a need for an acceleration technique in order to cut the link between accuracy and complexity of the value iteration.
For sake of clarity, the above framework has been presented for the infinite horizon optimal control problem. However, similar ideas can be extended to other classical control problems with small changes. Let us mention how to deal with the minimum time problem which we will use in the final section on numerical tests.
In the minimum time problem, one has to drive the controlled dynamical system (2.1) from its initial state to a given target $\mathcal{T}$. Let us assume that the target is a compact subset of $\mathbb{R}^d$ with non empty interior and piecewise smooth boundary. The major difficulty dealing with this problem is that the time of arrival to the target starting from the point x

$$t(x,\alpha(\cdot)) := \begin{cases} \inf\limits_{\alpha\in\mathcal{A}} \{t\in\mathbb{R}_+ : y(t,\alpha(\cdot))\in\mathcal{T}\} & \text{if } y(t,\alpha(t))\in\mathcal{T} \text{ for some } t, \\ +\infty & \text{otherwise,} \end{cases} \tag{2.8}$$

can be infinite at some points. As a consequence, the minimum time function defined as

$$T(x) = \inf_{\alpha\in\mathcal{A}} t(x,\alpha(\cdot)) \tag{2.9}$$

is not defined everywhere unless some controllability assumptions are introduced. In general, this is a free boundary problem where one has to determine at the same time, the couple (T,Ω), i.e. the minimum time function and its domain. Nevertheless, by applying the Dynamic Programming Principle and the so-called Kruzkhov transform

$$v(x) \equiv \begin{cases} 1-\exp(-T(x)) & \text{for } T(x)<+\infty \\ 1 & \text{for } T(x)=+\infty \end{cases} \tag{2.10}$$

the minimum time problem is characterized in terms of the unique viscosity solution of the BVP

$$\begin{cases} v(x) + \sup\limits_{a\in\mathcal{A}}\{-f(x,a)\cdot Dv(x)\} = 1 & \text{in } \mathcal{R}\backslash\mathcal{T} \\ v(x) = 0 & \text{on } \partial\mathcal{T}\,, \end{cases} \tag{2.11}$$

where $\mathcal{R}$ stands for the set of point in the state space where the time of arrival is finite. Then, the application of the semi-Lagrangian method presented for the infinite horizon optimal control problem together with a value iteration procedure leads to following iterative scheme:

Algorithm 2: Value Iteration for minimum time optimal control **(VI)**

Data: Mesh G, Δt, initial guess V^0, tolerance ϵ.
Set: $V_i = 0$, for all $x_i \in G\cap\mathcal{T}$
while $||V^{k+1}-V^k|| \geq \epsilon$ **do**
  forall the $x_i \in G\setminus\mathcal{T}$ **do**
    Solve: $V_i^{k+1} = \min\limits_{a\in A}\{e^{-\Delta t} I\left[V^k\right](x_i+\Delta t f(x_i,a)) + 1 - e^{-\Delta t}\}$ (2.12)
  end
  $k = k+1$
end

The numerical implementation is completed with the boundary conditions $v(x) = 0$ at $\partial\mathcal{T}$ (and inside the target as well), and with $v(x) = 1$ at other points outside the computational domain (we refer the reader to [6] for more details on the approximation of minimum time problems).

Policy iteration. We now turn our attention to an alternative solution method for discretized HJB equations of the form 2.6. The *approximation in the policy space* (or policy iteration), and is based on a linearization of the Bellman equation. First, an initial guess for the control for every point in the state space is chosen. Once the control has been fixed, the Bellman equation becomes linear (no search for the minimum in the control space is performed), and it is solved as an advection equation. Then, an updated policy is computed and a new iteration starts. Let us sketch the procedure for the scheme related to the infinite horizon problem.

Algorithm 3: Policy Iteration for infinite horizon optimal control **(PI)**

Data: Mesh G, Δt, initial guess V^0, tolerance ϵ.
while $||V^{k+1} - V^k|| \geq \epsilon$ **do**
  Policy evaluation step:
  forall the $x_i \in G$ **do**

$$V_i^k = \Delta t g\left(x_i, a_i^k\right) + (1 - \lambda\Delta t) I\left[V^k\right]\left(x_i + \Delta t f\left(x_i, a_i^k\right)\right) \tag{2.13}$$

  end
  Policy improvement step:
  forall the $x_i \in G$ **do**

$$a_i^{k+1} = \arg\min_a \left\{\Delta t g(x_i, a) + (1 - \lambda\Delta t) I\left[V^k\right](x_i + \Delta t f(x_i, a))\right\} \tag{2.14}$$

  end
  $k = k + 1$
end

Note that the solution of (2.13) can be obtained either by a linear system (assuming I is a linear interpolation operator over the dataset V) or as the limit

$$V^k = \lim_{m \to +\infty} V^{k,m}, \tag{2.15}$$

of the linear time-marching scheme

$$V_i^{k,m+1} = \Delta t g\left(x_i, a_i^k\right) + (1 - \lambda\Delta t) I\left[V^{k,m}\right]\left(x_i + \Delta t f\left(x_i, a_i^k\right)\right). \tag{2.16}$$

Although this scheme is still iterative, the lack of a minimization phase makes it faster than the original value iteration.

The sequence $\{V^k\}$ turns out to be monotone decreasing at every node of the grid. In fact, by construction,

$$\begin{aligned} V_i^k &= \Delta t g\left(x_i, a_i^k\right) + (1 - \lambda\Delta t) I\left[V^k\right]\left(x_i + \Delta t f\left(x_i, a_i^k\right)\right) \geq \\ &\geq \min_a \left\{\Delta t g(x_i, a) + (1 - \lambda\Delta t) I\left[V^k\right](x_i + \Delta t f(x_i, a))\right\} = \\ &= \Delta t g\left(x_i, a_i^{k+1}\right) + (1 - \lambda\Delta t) I\left[V^k\right]\left(x_i + \Delta t f\left(x_i, a_i^{k+1}\right)\right) = \\ &= V_i^{k+1} \end{aligned}$$

At a theoretical level, policy iteration can be shown to be equivalent to a Newton method, and therefore, under appropriate assumptions, it converges with quadratic

speed. On the other hand, convergence is local and this may represent a drawback with respect to value iterations.

3. An accelerated policy iteration algorithm with smart initialization. In this section we present an accelerated iterative algorithm which is constructed upon the building blocks previously introduced. We aim at an efficient formulation exploiting the main computational features of both value and policy iteration algorithms. As it has been stated in [26], there exists a theoretical equivalence between both algorithms, which guarantees a rather wide convergence framework. However, from a computational perspective, there are significant differences between both implementations. A first key factor can be observed in Figure 3.1 which shows, for a two-dimensional minimum time problem (more details on the test can be found in section 4.4), the typical situation arising with the evolution of the error measured with respect to the optimal solution, when comparing value and policy iteration algorithms. To achieve a similar error level, policy iteration requires considerable fewer iterations than the value iteration scheme, as quadratic convergent behavior is reached faster for any number of nodes in the state-space grid. Despite the observed computational

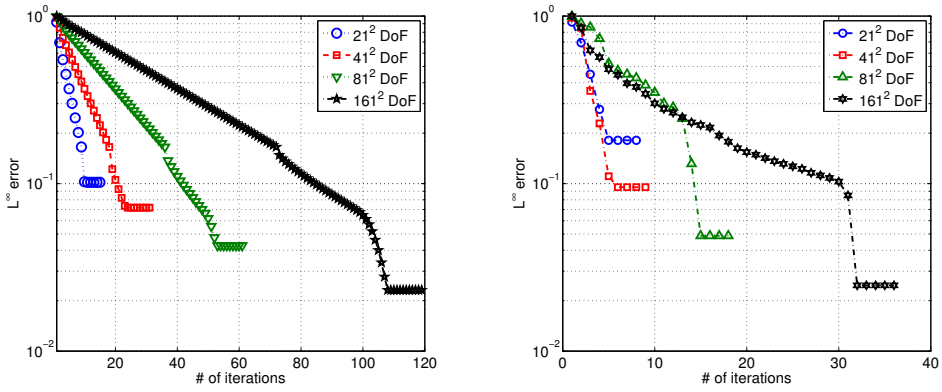

FIG. 3.1. *Error evolution in a 2D problem: value iteration (left) and policy iteration (right).*

evidence, a second issue is observed when examining the policy iteration algorithm in more detail. That is, as shown in Figure 3.2, the sensitivity of the method with respect to the choice of the initial guess of the control field. It can be seen that different initial admissible control fields can lead to radically different convergent behaviors. While some guesses will produce quadratic convergence from the beginning of the iterative procedure, others can lead to an underperformant value iteration-like evolution of the error. This latter is computationally costly, because it translates into a non-monotone evolution of the subiteration count of the solution of equation (2.13) (if an iterative scheme is used as in (2.16)).

A final relevant remark goes back to Figure 3.1, where it can be observed that for coarse meshes, the value iteration algorithm generates a fast error decay up to a higher global error. This, combined with the fact that value iteration algorithms are rather insensitive to the choice of the initial guess for the value function (see [27] for a detailed error quantification), are crucial points for the construction of our accelerated algorithm. The accelerated policy iteration algorithm is based on a robust initialization of the policy iteration procedure via a coarse value iteration which will

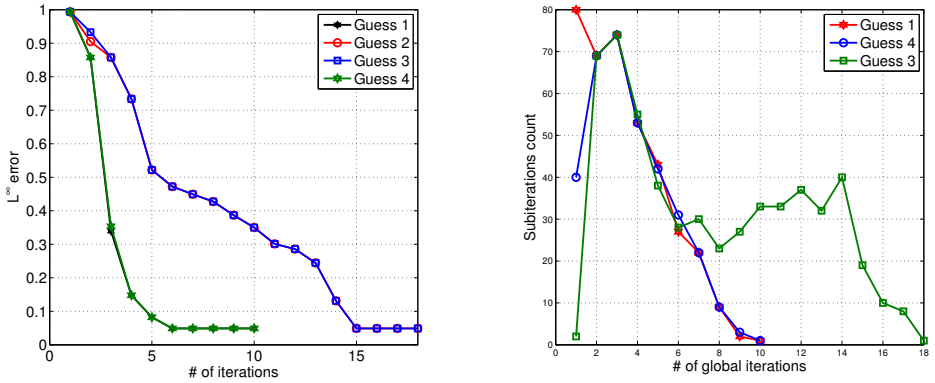

FIG. 3.2. *Left: error evolution in a PI algorithm for different initial guesses. Right: evolution of the (sub)iteration count in* (2.16) *for different guesses.*

yield to a good guess of the initial control field.

Algorithm 4: Accelerated Policy Iteration **(API)**

Data: Coarse mesh G_c and Δt_c , fine mesh G_f and Δt_f, initial coarse guess V_c^0, coarse-mesh tolerance ϵ_c, fine-mesh tolerance ϵ_f.

begin

Coarse-mesh value iteration step: perform Algorithm 1

Input: G_c, Δt_c, V_c^0, ϵ_c

Output: V_c^*

forall the $x_i \in G_f$ **do**

$V_f^0(x_i) = I_1[V_c^*](x_i)$

$A_f^0(x_i) = \underset{a \in A}{argmin} \{(1 - \lambda \Delta t) I_1[V_f^0](x_i + f(x_i, a)) + \Delta t g(x_i, a)\}$

end

Fine-mesh policy iteration step: perform Algorithm 3

Input: G_f, Δt_f, V_f^0, A_f^0, ϵ_f

Output: V_f^*

end

3.1. Practical details concerning the computational implementation of the algorithm. The above presented accelerated algorithm can lead to a considerably improved performance when compared to value iteration and naively initialized policy iteration algorithms. However, it naturally contains trade-offs that need to be carefully handled in order to obtain a correct behavior. The extensive numerical tests performed in Section 4 suggest the following guidelines:

Coarse and fine meshes. The main trade-off of the accelerated algorithm is related to this point. For a good behavior of the PI part of the algorithm, a good initialization is required, but this should be obtained without deteriorating the overall performance. Too coarse VI will lead to poor initialization, whereas fine VI will increase the CPU time. We recall that for this paper we assume regular equidistant meshes with mesh parameter Δx. If we denote by Δx_c and by Δx_f the mesh parameters associated to the coarse and fine grids respectively, numerical findings illustrated in Figure 3.3

suggest that for minimum time problems and infinite horizon optimal control, a good balance is achieved with $\Delta x_c = 2\Delta x_f$. In the case of minimum time problems, it is also important that the coarse mesh is able to accurately represent the target.

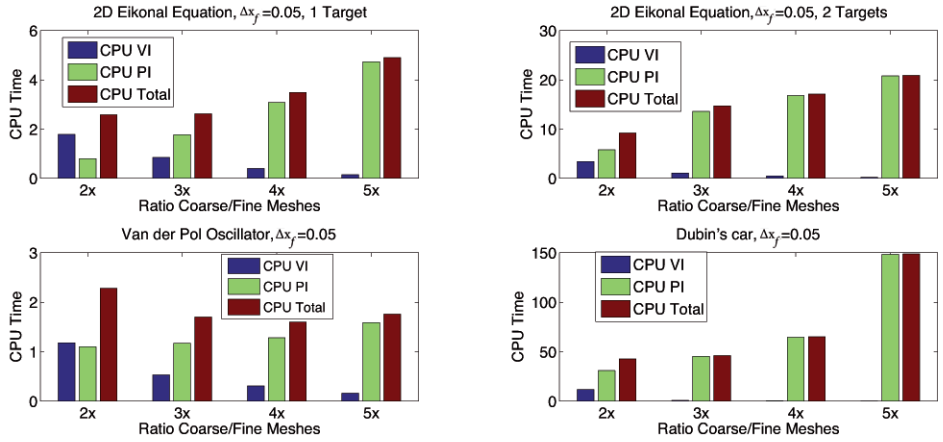

FIG. 3.3. *Ratios $\Delta x_c/\Delta x_f$ and CPU time for different control problems. A good overall balance can be observed in most cases by considering $\Delta x_c = 2\Delta x_f$.*

Accuracy. Both VI and PI algorithms require a stopping criterion for convergence. Following [29], the stopping rule is given by

$$||V^{k+1} - V^k|| \leq C\Delta x^2\,,$$

which relates the error to the resolution of the state-space mesh. The constant C is set to $C = \frac{1}{5}$ for the fine mesh, and for values ranging from 1 to 10 in the coarse mesh, as we do not strive for additional accuracy that usually will not improve the initial guess of the control field. However, different options have been extensively discussed in the literature, as in [28] for instance, where the stopping criteria is related to a variability threshold on the control space.

Policy evaluation. In every main cycle of the policy iteration algorithm, provided the interpolation operator is linear, as it is in our case, a solution of the linear system (2.13) is required. This can be performed in several ways, specially given the sparsity of the system. For sake of simplicity and in order to make numerical comparisons with the VI scheme, we use a fixed point iteration, i.e., the policy evaluation is implemented as

$$V_i^{k,j+1} = \Delta t g\left(x_i, a_i^k\right) + (1 - \lambda\Delta t) I\left[V^{k,j}\right]\left(x_i + \Delta t f\left(x_i, a_i^k\right)\right) \tag{3.1}$$

with initial guess $V^{k,0} = V^{k-1,\infty}$. We use the same stopping criteria as for the global iteration.

Minimization. Although counterexamples can be constructed in order to show that it is not possible to establish error bounds of the PI algorithm independently of the (finite) number of controls [29], the algorithm does not change its performance when the control set is increased, and therefore the *argmin* computation required for the policy update can be performed by discretizing the set of controls and evaluating all the possible arrival points. Note that, in order to avoid the discretization of the control set, minimizers can be computed using Brent's algorithm, as in [12].

A remark on parallelism. Although the numerical tests that we present were performed in a serial code, we note that the accelerated algorithm allows an easy parallel implementation. Whenever an iterative procedure is performed over the value function, parallelism can be implemented via a domain decomposition of the state space as in [16,9]. If the control space is also discretized, the policy update (2.14) can also be parallelized with respect of the set of controls.

4. Numerical tests. This section presents a comprehensive set of tests assessing the performance of the proposed accelerated algorithm. We compare the results with solutions given by the classical value iteration algorithm, policy iteration, and the accelerated monotone value iteration method. In some examples we also include an accelerated algorithm based on a monotone value iteration in the set of subsolutions (AMVI), as presented in [4, Appendix A], and a Gauss-Seidel variation of this method (GSVI) as in [18]. In a first part we develop tests related to infinite horizon optimal control, to then switch to the study of minimum time problems. We conclude with an extension to applications related to optimal control of partial differential equations. We focus on grid resolution, size of the discretized control space, performance in presence of linear/nonlinear dynamics, targets, and state space dimension. All the numerical simulations reported in this paper have been made on a MacBook Pro with 1 CPU Intel Core i5 2.3 Ghz and 8GB RAM.

Infinite horizon optimal control problems

4.1. Test 1: A non-smooth 1D value function. We first consider a one-dimensional optimal control problem appearing in [4, Appendix A]. Using a similar notation as in Section 2, we set the computational domain $\Omega =]-1\,,1[$, the control space $A = [-1\,,1]$, the discount factor $\lambda = 1$, the system dynamics $f(x,a) = a(1-|x|)$, and the cost function $g(x,a) = 3(1-|x|)$. The exact optimal solution for this problem is

$$v(x) = \begin{cases} \frac{3}{2}(x+1) & \text{for } x < 0\,, \\ \frac{3}{2}(1-x) & \text{elsewhere}\,, \end{cases}$$

which has a kink at $x = 0$. We implement every proposed algorithm, and results concerning CPU time and number of iterations are shown in Table 4.1; for different mesh configurations, we set $\Delta t = .5\Delta x$ and we discretize the control space into a set of 20 equidistant points. The notation VI($2\Delta x$) in Table 4.1 stands for the computation of the solution with a VI method considering a coarse grid of $2\Delta x$. Then it is applied the PI method with a stepsize Δx. (PI(Δx) in the table). This notation, in the table, is kept in all the tests. In this test case, as expected, we observe that the VI algorithm is always the slowest option, with iteration count depending on the number of mesh nodes; this feature is also observed for the PI algorithm, although the number of iterations and CPU time are considerably smaller. On the other hand, the AMVI scheme has an iteration count independent of the degrees of freedom of the system, with an almost fixed CPU time, as the time spent on fixed point iterations is negligible compared to the search of the optimal update direction. In this particular example, the exact boundary conditions of the problem are known ($v(x) = 0$ at $\partial\Omega$) and it is possible to construct monotone iterations by starting from the initial guess $v(x) = 0$. The GSVI method exhibits a similar performance as the PI algorithm, with a considerably reduced number of iterations when compared to VI. Note however, that this implementation requires the pre-computation and storage of

the interpolation coefficients and a sequential running along the mesh, which can be unpractical of high-dimensional problems are considered. Finally, the API algorithm exhibits comparable CPU times as AMVI, performing always better than VI, PI and GSVI. In this particular case, the choice of the mesh ratio between the coarse and fine meshes can be suboptimal, as the time spent on the VI coarse pre-processing represents an important part of the overall CPU time. More details on the error evolution throughout the iterations can be observed in Figure 4.1; note that the error evolution is measured with respect to the exact solution and not with respect to the next iteration. This latter figure illustrates, for both problems, the way in which the API idea acts: pre-processing of the initial guess of PI leads to proximity to a quadratic convergence neighborhood, where this algorithm can converge in a reduced number of iterations; the fast error decay that coarse mesh VI has in comparison with the fine mesh VI is clearly noticeable. In Table 4.2, we show the performance evolution of the different algorithms when the parameter λ decreases. It is expected that for methods based on a fixed point iteration of the value function, the number of iterations required to reach a prescribed error level will gradually increase. This is clearly observed for VI, PI and API, whereas AMVI and GSVI are able to circumvent this difficulty, leading to a constant number of iterations independent of the parameter λ. Nevertheless, in the overall cpu time, GSVI and API exhibit a similar asymptotic performance.

# nodes	Δx	VI	PI	AMVI	GSVI	VI($2\Delta x$)	PI(Δx)	API
81	2.5E-2	9.88E-2 (228)	2.02E-2 (10)	1.99E-2 (3)	2.25E-2 (41)	5.31E-3(23)	5.22E-3 (2)	1.05E-2
161	1.25E-2	0.41 (512)	5.88E-2 (34)	3.8E-2 (3)	7.71E-2 (81)	3.21E-2(73)	1.73E-2 (2)	4.94E-2
321	6.25E-3	1.89 (1134)	0.21 (65)	7.48E-2 (3)	0.29 (161)	0.16(200)	2.62E-2 (2)	0.19

TABLE 4.1
Test 1 (1D non-smooth value function): CPU time (iterations) for different algorithms.

λ	VI	PI	AMVI	GSVI	VI($2\Delta x$)	PI(Δx)	API
1	1.31 (1134)	0.16 (65)	5.73E-2 (3)	0.19 (161)	7.41E-2 (112)	3.20E-2 (2)	0.11
0.1	2.45 (2061)	0.46 (138)	5.82E-2 (3)	0.19 (161)	0.12 (203)	5.41E-2 (2)	0.18
1E-2	2.63 (2244)	0.67 (159)	6.18E-2 (3)	0.19 (161)	0.12 (220)	6.483E-2 (2)	0.19
1E-3	2.65 (2265)	0.74 (161)	7.75E-2 (4)	0.19 (161)	0.13 (222)	6.41E-2 (2)	0.19

TABLE 4.2
Test 1 (1D non-smooth value function): CPU time (iterations) for different algorithms and different values of λ, in a fixed mesh with 321^2 nodes and 2 control values.

4.2. Test 2: Van Der Pol oscillator. In a next step we consider two-dimensional, nonlinear system dynamics given by the Van der Pol oscillator:

$$f(x,y,a) = \begin{pmatrix} y \\ (1-x^2)y - x + a \end{pmatrix}.$$

System parameters are set:

$$\Omega =]-2\,,2[^2\,, \quad A = [-1\,,1]\,, \quad \lambda = 1\,, \quad \Delta t = 0.3\Delta x\,, \quad g(x,y,a) = x^2 + y^2\,,$$

and the control space is discretized into 32 equidistant points. We perform a similar numerical study as in the previous example, and results are shown in Table 4.3.

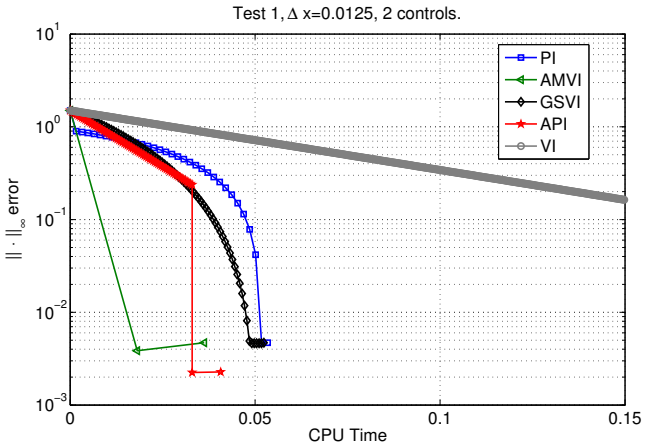

FIG. 4.1. *Test 1 (non-smooth value function): error evolution for different algorithms.*

For computations requiring an exact solution, we consider as a reference a fine grid simulation with $\Delta x = 6.25E-3$.

We set a constant boundary value $v(x) = 3.5$ at $\partial\Omega$, which can be interpreted as a penalization on the state. If accurate solutions near the boundary are required, a natural choice in our setting would be to perform simulations over an enlarged domain and then restrict the numerical results to a subset of interest. From this test we observe a serious limitation on the AMVI algorithm. The number of iterations now depends on the number of nodes, and even though the number of iterations is still lower than in the VI algorithm, the CPU time increases as for every iteration a search procedure is required. As it is not possible to find monotone update directions, the AMVI algorithm becomes a VI method plus an expensive search procedure. This lack of possible monotone update can be due to several factors: the nonlinear dynamics, the existence of trajectories exiting the computational domain, and a sensitivity to the artificial boundary condition. We report having performed similar tests for the linear double integrator problem ($\ddot{x} = a$) with similar results, therefore we conjecture that in this case, the underperformance of the AMVI scheme is due to poor boundary resolution and its use by optimal trajectories. Unfortunately, this is a recurrent problem in the context of optimal control. This situation does not constitute a problem for the API algorithm, where a substantial speedup is seen in both coarse and fine meshes. Note that compared to PI, the accelerated scheme has a number of iterations on its second part which is independent of the mesh parameters as we are in a close neighborhood of the optimal solution.

# nodes	Δx	VI	PI	AMVI	VI($2\Delta x$)	PI(Δx)	API
81^2	5E-2	39.6 (529)	5.35 (8)	1.42E2 (3)	1.86 (207)	1.47 (4)	3.33 (211)
161^2	2.5E-2	3.22E2 (1267)	34.5 (11)	1.01E3 (563)	10.7(165)	6.87 (4)	17.5 (169)
321^2	1.25E-2	3.36E4 (2892)	3.36E2 (14)	1.55E4 (2247)	88.9 (451)	47.7 (4)	1.36E2 (455)

TABLE 4.3
Test 2 (Van der Pol oscillator): CPU time (iterations) for different algorithms.

4.3. Test 3: Dubin's Car. Having tested some basic features of the proposed schemes, we proceed with our numerical study of the API method by considering a three-dimensional nonlinear dynamical system given by

$$f(x,y,z,a) = \begin{pmatrix} \cos(z) \\ \sin(z) \\ a \end{pmatrix},$$

corresponding to a simplified version of the so-called Dubin's car, a test problem extensively used in the context of reachable sets and differential games. System parameters are set:

$$\Omega =]-2\,,2[^2\,, \quad A = [-1\,,1]\,, \quad \lambda = 1\,, \quad \Delta t = 0.2\Delta x\,, \quad g(x,y,z,a) = x^2 + y^2\,,$$

and the control space is discretized into 11 equidistant points; the boundary value is set to $v(x) = 3$ in $\partial\Omega$ and reference solution is taken with $\Delta x = 1.25E-2$. Different isosurfaces for this optimal control problem can be seen in Figure 4.2, and CPU times for different meshes are shown in Table 4.4. This case is an example in which the mesh ratio between coarse and fine meshes is well-balanced, and the time spent in pre-processing via VI is not relevant in the overall API CPU time, despite leading to a considerable speedup of the order of $8\times$ with a mesh of 10^6 grid points. In the last line of Table 4.4, the VI algorithm was stopped after 4 hours of simulation without achieving convergence, which is illustrative of the fact that acceleration techniques in such problems are not only desirable but necessary in order to obtain results with acceptable levels of accuracy.

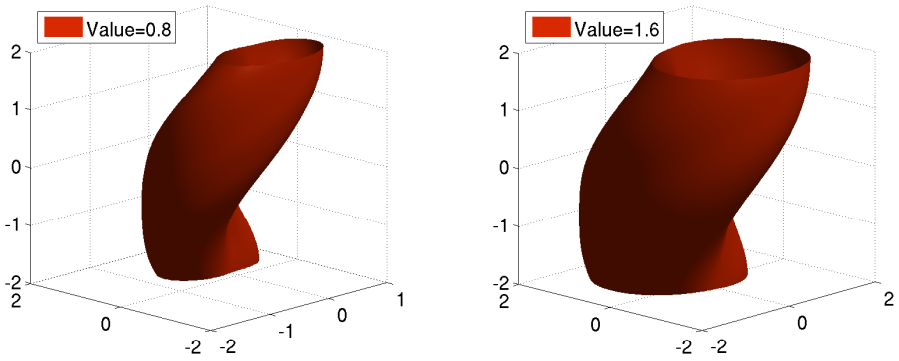

FIG. 4.2. *Test 3: Dubin's car value function isosurfaces.*

# nodes	Δx	VI	PI	VI($2\Delta x$)	PI(Δx)	API
41^3	0.1	50.6 (192)	12.2 (12)	0.84 (8)	8.52 (3)	9.36 (11)
81^3	5E-2	1.19E3 (471)	3.28E2 (18)	8.98 (39)	1.39E2 (9)	1.48E2 (48)
161^3	2.5E-2	$\geq$ 1.44E4	9.93E3 (12)	3.02E2 (30)	2.92E3 (10)	2.62E3 (40)

TABLE 4.4
Test 3 (Dubin's car): CPU time (iterations) for different algorithms

Minimum time problems

4.4. Tests 4 and 5: minimum time problems in 2D. The next two cases are based on a two-dimensional eikonal equation. For both problems, common seetings are given by

$$f(x,y,a) = \begin{pmatrix} \cos(a) \\ \sin(a) \end{pmatrix}, \quad A = [-\pi,\, \pi], \quad \Delta t = 0.8\Delta x\,.$$

What differentiates the problems is the domain and target definitions; Test 4 considers a domain $\Omega =]-1,\, 1[^2$ and a target $\mathcal{T} = (0,0)$, while for Test 5, $\Omega =]-2,\, 2[^2$ and $\mathcal{T} = \{x \in \mathbb{R}^2 \,:\, ||x||_2 \leq 1\}$. Reference solutions are considered to be the distance function to the respective targets, which is an accurate approximation provided that the number of possible control directions is large enough. For Test 4, with a discretization of the control space into set of 64 equidistant points, CPU time results are presented in Table 4.5; it can be seen that API provides a speedup of $8\times$ with respect to VI over fine meshes despite the large set of discrete control points. Table 4.6 shows experimental convergence rates achieved by the fully discrete scheme, in both L^1 and L^∞ norms, which are in accordance with the theoretically expected rate of $1/2$. Test 5 features an enlarged target, and differences in CPU time are presented in Table 4.7 where, for a discrete set of 72 equidistant controls, the speedup is reduced to $4\times$. In general, from a mesh node, larger or more complicated targets represent a difficulty in terms of the choice of the minimizing control, which translates into a larger number of iterations. In this case, the CPU time spent in the pre-processing is significant to the overall CPU time, but increasing this ratio in order to reduce its share will lead to an underperformant PI part of the algorithm.

# nodes	Δx	VI	PI	VI($2\Delta x$)	PI(Δx)	API
41^2	5E-2	3.16 (37)	1.89 (12)	0.39 (5)	0.38 (2)	0.77 (7)
81^2	2.5E-2	8.23 (69)	4.43 (19)	0.80 (12)	0.53 (2)	1.33 (14)
161^2	1.25E-2	39.2 (133)	12.6 (13)	2.55 (31)	2.11 (3)	4.66 (34)

TABLE 4.5
Test 4 (2D eikonal): CPU time (iterations) for different algorithms.

# nodes	Δx	$L^1 - error$	rate	$L^\infty - error$	rate
41^2	5E-2	2.1E-2	0.60	8.9E-3	0.61
81^2	2.5E-2	1.4E-2	0.64	5.8E-3	0.64
161^2	1.25E-2	8.5E-3	0.68	3.7E-3	0.75
321^2	6.25E-3	5.3E-3		2.2E-3	

TABLE 4.6
Test 4 (2D Eikonal): Rate of convergence for the API scheme with 64 controls.

4.5. Tests 6 and 7: minimum time problems in 3D. We develop a three-dimensional extension of the previously presented examples. System dynamics and common parameters are given by

$$f(x,y,z,(a_1,a_2)) = \begin{pmatrix} \sin(a_1)\cos(a_2) \\ \sin(a_1)\sin(a_2) \\ \cos(a_1) \end{pmatrix}, \quad A = [-\pi,\, \pi] \times [0\,, \pi]\,, \quad \Delta t = 0.8\Delta x\,.$$

# nodes	Δx	VI	PI	VI($2\Delta x$)	PI(Δx)	API
64^2	6.35E-2	4.02 (36)	1.42 (9)	0.84 (10)	0.53 (4)	1.37 (14)
128^2	3.15E-2	16.9 (70)	6.25 (14)	2.80 (25)	1.66 (2)	4.46 (27)
256^2	1.57E-2	1.09E2 (135)	38.7 (16)	15.8 (62)	11.7 (8)	27.5 (70)
512^2	7.8E-3	9.80E2 (262)	3.98E2(168)	1.07E2 (126)	1.09E2 (12)	2.16E2 (138)

TABLE 4.7
Test 5 (2D Eikonal): CPU time (iterations) for different algorithms with 72 controls.

As in the two-dimensional study, we perform different tests by changing the domain and the target. For Test 6 we set $\Omega =]-1, 1[^3$ and $\mathcal{T} = (0, 0, 0)$, while for Test 7, $\Omega =]-6, 6[$ and $\mathcal{T}$ is the union of two unit spheres centered at $(-1, 0, 0)$ and $(1, 0, 0)$. In both cases, the set of controls is discretized into 16×8 points. Reachable sets for Test 7 are shown in Figure 4.3, and CPU times for both tests can be found in Tables 4.8 and 4.9. We observe similar results as in the 2D tests, with up to $10\times$ acceleration for a simple target, and $4\times$ with more complicated targets. Note that in the second case, the speedup is similar to the natural performance that would be achieved by a PI algorithm. This is due to a weaker influence of the coarse VI iteration, which is sensitive to poor resolution of a complex target.

# nodes	Δx	VI	PI	VI($2\Delta x$)	PI(Δx)	API
41^3	5E-2	4.83E2 (44)	1.22E3 (10)	4.61 (5)	1.19E2 (3)	1.23E2 (8)
81^3	2.5E-2	7.67E3 (84)	1.47E3 (13)	2.43E1 (12)	3.88E2 (3)	6.31E2 (15)

TABLE 4.8
Test 6 (3D Eikonal): CPU time (iterations) for different algorithms with $a_1 = 16$ controls, $a_2 = 8$ controls.

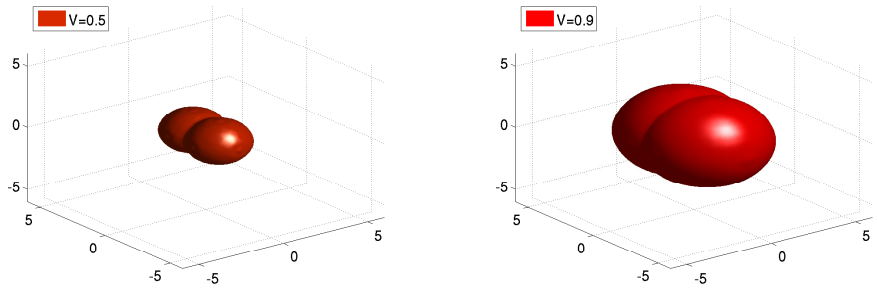

FIG. 4.3. *Test 7 (3D eikonal): different value function isosurfaces.*

4.6. Test 8: A Minimum time Problem in 4D. We conclude our series of tests in minimum time problems by considering a four-dimensional problem with a relatively reduced control space. In the previous examples we have studied the performance of our scheme in cases where the set of discrete controls was fairly large, while in several applications, it is also often the case that the set of admissible discrete controls is limited and attention is directed towards the dimensionality of the state

# nodes	Δx	VI	PI	VI($2\Delta x$)	PI(Δx)	API
61^3	0.2	2.67E2 (25)	1.22E2 (9)	1.44 (11)	6.80E1 (3)	6.94E1 (14)
121^3	0.1	4.52E3 (52)	1.28E3 (11)	25.15E1 (12)	9.96E2 (3)	1.01E3 (15)

TABLE 4.9
Test 7 (3D Eikonal): CPU time (iterations) for different algorithms with $a_1 = 16$ controls, $a_2 = 8$ controls.

space. The following problem tries to mimic such a setting. System dynamics are given by

$$f(x, y, z, w, (a_1, a_2, a_3, a_4)) = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix},$$

the domain is $\Omega =]-1, 1[^4$, the target is $\mathcal{T} = \partial\Omega$, $\Delta t = 0.8\Delta x$ and A is the set of 8 directions pointing to the facets of the four-dimensional hypercube. Figure 4.4 shows different reachable sets and CPU times are presented in Table 4.10. In the finest mesh a speedup of 8× is observed, which is consistent with the previous results on simple targets. Thus, the performance of the presented algorithm is not sensitive neither to the number of discrete controls nor to the dimension of the state space, whereas it is affected by the complexity of the target.

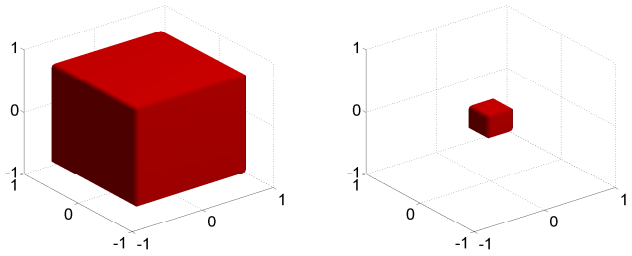

FIG. 4.4. *Test 8 (4D minimum time): different value function isosurfaces with $x_4 = 0$.*

# nodes	Δx	VI	PI	VI($2\Delta x$)	PI(Δx)	API
21^4	0.1	13.6 (15)	16.2 (11)	0.30 (4)	2.79 (2)	3.09 (6)
41^4	5E-2	4.79E2 (29)	6.30E2 (21)	10.2 (12)	48.3 (2)	58.5 (14)

TABLE 4.10
Test 8 (4D minimum time): CPU time (iterations) for different algorithms.

4.7. Application to optimal control problem of PDEs. Having developed a comprehensive set of numerical tests concerning the solution of optimal control problems via static HJB equations, which assessed the performance of the proposed API algorithm, we present an application where the existence of accelerated solution techniques for high-dimensional problems is particularly relevant, namely, the optimal control of systems governed by partial differential equations. From an abstract

perspective, optimal control problems where the dynamics are given by evolutive partial differential equations correspond to systems where the state lies in an infinite-dimensional Hilbert space (see [33]). Nevertheless, in terms of practical applications, different discretization arguments can be used to deal with this fact, and (sub)optimal control synthesis can be achieved through finite-dimensional, large-scale approximations of the system. At this step, the resulting large-scale version will scale according to a finite element mesh parameter, and excepting for the linear-quadratic case and some closely related versions, it would be still computationally intractable for modern architectures (for instance, for a 100 elements discretization of a 1D PDE, the resulting optimal control would be characterized as the solution of a HJB equation in $\mathbb{R}^{100}$). Therefore, a standard remedy in optimal control and estimation is the application of model order reduction techniques, which, upon a large-scale version of the system, recover its most relevant dynamical features in a low-order approximation of prescribed size. In this context, surprisingly good control synthesis can be achieved with a reduced number of states (for complex nonlinear dynamics and control configurations an increased number of reduced states may be required). Previous attempts in this direction dates back to [23,24] and more recently to [1,2]. We present an example where we embed our accelerated algorithm inside the described framework. Note that, in this example, model reduction method is only applied in order to make the problem feasible for the Dynamic Programming approach. The acceleration is due to the proposed API scheme.

Let us consider a minimum time problem for the linear heat equation:

$$\begin{cases} y_t(x,t) = c y_{xx}(x,t) + y_0(x)\alpha(t)\,, \\ y(0,t) = y(1,t) = 0\,, \\ y(x,0) = y_0(x)\,, \end{cases} \tag{4.1}$$

where $x \in [0,1]$, $t \in [0,T]$, $c = 1/80$ and $\alpha(t) : [0,T] \to \{-1,0,1\}$. After performing a finite difference discretization, we perform a Galerkin projection with basis function computed with a Proper Orthogonal Decomposition (POD) method, leading to a reduced order model (we refer to [34] for an introduction to this topic). In general, model reduction techniques do have either a priori or a posteriori error estimates which allow to prescribe a certain number of reduced states yielding a desired level of accuracy. For this simple case, we consider the first 3 reduced states, which for a one-dimensional heat transfer process with one external source provides a reasonable description of the input-output behavior of the system. The system is reduced to:

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -0.123 & -0.008 & -0.001 \\ -0.008 & -1.148 & -0.321 \\ -0.001 & -0.321 & -3.671 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} -5.770 \\ -0.174 \\ -0.022 \end{bmatrix}\alpha(t). \tag{4.2}$$

Once the reduced model has been obtained, we solve the minimum time problem with target $\mathcal{T} = (0,0,0)$. Figure 4.5 shows contour plots of the value function in the reduced space and a comparison of the performance of the minimum time controller with respect to the uncontrolled solution and to a classical linear-quadratic controller is presented. CPU times are included in Table 4.11, where a speedup of $4\times$ can be observed, the acceleration would become more relevant as soon as more refined meshes and complex control configurations are considered.

Concluding remarks and future directions. In this work we have presented an accelerated algorithm for the solution of static HJB equations arising in different

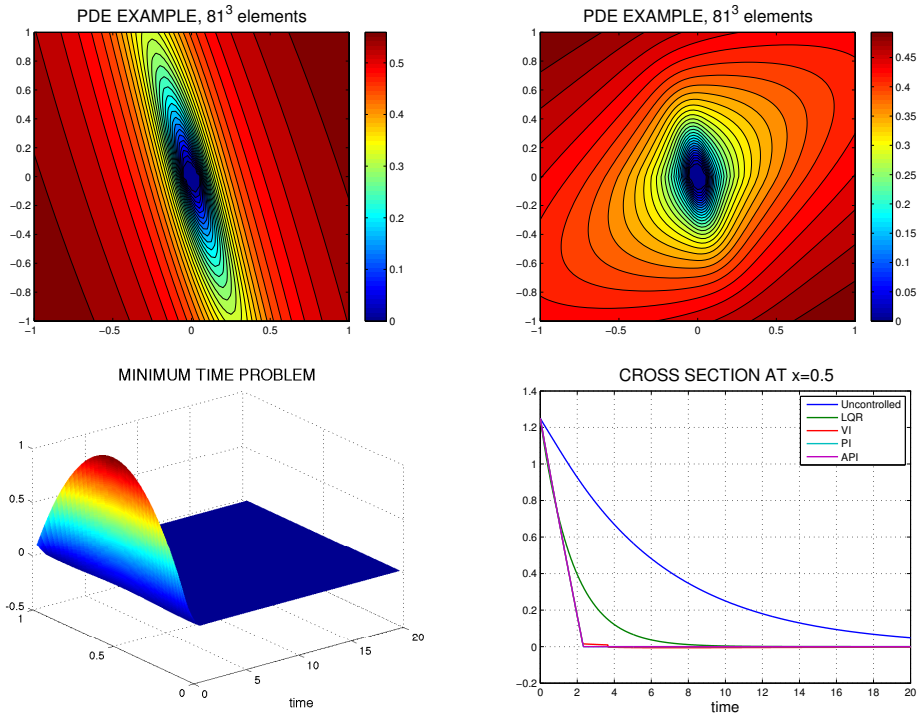

FIG. 4.5. *Optimal control of the heat equation. Top left: contour plot of the value function at* $x_3 = 0$. *Top right: contour plot of the value function at* $x_2 = 0$. *Bottom left: controlled output via proposed procedure of model reduction + minimum time HJB controller. Bottom right: cross section of the different outputs.*

# nodes	Δx	VI	PI	VI($2\Delta x$)	PI(Δx)	API
21^3	0.1	1.87 (76)	0.91 (11)	0.32 (27)	0.59 (8)	0.98 (35)
41^3	5E-2	27.8 (178)	12.4 (15)	1.65 (76)	6.34 (10)	7.99 (86)
81^3	2.5E-2	6.13E2 (394)	2.68E2 (15)	27.7 (178)	1.45E2 (9)	1.72E2 (187)

TABLE 4.11
Minimum time control of the heat equation: CPU time (iterations) for different algorithms.

optimal control problems. The proposed method considers a pre-processing value iteration procedure over a coarse mesh with relaxed stopping criteria, which is used to generate a good initial guess for a policy iteration algorithm. This leads to accelerated numerical convergence with respect to the known approximation methods, with a speedup ranging in average from $4\times$ to $8\times$. We have assessed the performance of the new scheme via a extensive set of numerical tests focusing on infinite horizon and minimum time problems, providing numerical evidence of the reliability of the method in tests with increasing complexity. Positive aspects of the proposed scheme are its wide applicability spectrum (in general for static HJB), and its insensitivity with respect to the complexity of the discretized control set. Nonetheless, for some non trivial targets, special care is needed in order to ensure that the coarse pre-processing step will actually lead to an improved behavior of the policy iteration scheme. Certainly, several directions of research remain open. The aim of this article was to present the

numerical scheme and provide a numerical assessment of its potential. Future works should focus on tuning the algorithm in order to achieve an optimal performance; for instance, in order to make a fair comparison with the value iteration algorithm, the policy iteration step was also performed via a successive approximation scheme, while better results could be obtained by using a more efficient solver, including a larger amount of pre-processing work. Other possible improvements would relate to multigrid methods, high-order schemes and domain decomposition techniques. An area of application that remains unexplored is the case of differential games, where Hamilton-Jacobi-Isaacs equations need to be solved. Results presented in [7] indicate that the extension is far from being trivial since a convergence framework is not easily guaranteed and the policy iteration scheme requires some modifications.

REFERENCES

[1] A. Alla and M. Falcone, *An adaptive POD approximation method for the control of advection-diffusion equations,* In Control and Optimization with PDE Constraints, K. Kunisch, K. Bredies, C. Clason, G. von Winckel (eds), Internat. Ser. of Numer. Math., Birkhäuser, Basel, **164** (2013), pp. 1–17.

[2] A. Alla and M. Falcone, *A Time-Adaptive POD Method for Optimal Control Problems*. Proceedings of the 1st IFAC Workshop on Control of Systems Modeled by Partial Differential Equations, 2013, pp. 245–250.

[3] K. Alton and I. M. Mitchell, *An ordered upwind method with precomputed stencil and monotone node acceptance for solving static convex Hamilton-Jacobi equations*, J. Sci. Comput., **51** (2012), pp. 313–348.

[4] M. Bardi and I. Capuzzo Dolcetta, Optimal control and viscosity solutions of Hamilton-Jacobi-Bellman equations, Birkhäuser, Basel, 1997.

[5] R. Bellman, Dynamic Programming. Princeton University Press, Princeton, NJ, 1957.

[6] M. Bardi and M. Falcone, *An approximation scheme for the minimum time function*, SIAM J. Control Optim., **28** (1990), pp. 950–965.

[7] O. Bokanowski, S. Maroso and H. Zidani, *Some convergence results for Howard's algorithm*, SIAM J. Numer. Anal. **47** (2009), pp. 3001–3026.

[8] N.D. Botkin, K. Hoffman and V. Turova, *Stable numerical schemes for solving Hamilton-Jacobi-Bellman-Isaacs equations*, SIAM J. Sci. Comput., **33** (2011), pp. 992-1007.

[9] S. Cacace, E. Cristiani. M. Falcone and A. Picarelli, *A patchy dynamic programming scheme for a class of Hamilton–Jacobi–Bellman equations*, SIAM J. Sci. Comput., **34** (2012), pp. 2625–2649.

[10] S. Cacace, E. Cristiani and M. Falcone, *Can local single pass methods solve any stationary Hamilton-Jacobi-Bellman equations?*, SIAM J. Sci. Comput., **36** (2014), pp. A570-A587.

[11] I. Capuzzo Dolcetta and M. Falcone, *Discrete dynamic programming and viscosity solutions*, Annales de l'Institut Henry Poincaré - Analyse non-linéaire, **6** (1989), pp. 161–184.

[12] E. Carlini, M. Falcone and R. Ferretti, *An efficient algorithm for Hamilton-Jacobi equations in high dimension*, Comput. Vis. Sc., **7** (2004), pp. 15–29.

[13] C. Chow and J.N. Tsitsiklis, *An optimal one-way multigrid algorithm for discrete-time stochastic control*, IEEE Trans. Automat. Control, **36** (1991), pp. 898–914.

[14] M. Falcone, *A numerical approach to the infinite horizon problem of deterministic control theory*, Appl. Math. Optim., **15** (1987), pp. 1–13.

[15] M. Falcone and R. Ferretti, Semi-Lagrangian Approximation Schemes for Linear and Hamilton-Jacobi Equations, SIAM, Philadelphia, 2014.

[16] M. Falcone, P. Lanucara and A. Seghini, *A splitting algorithm for Hamilton-Jacobi-Bellman equations*, Appl. Numer. Math., **15** (1994), pp. 207–218.

[17] W. H. Fleming and R.W. Rishel, Deterministic and stochastic optimal control, Springer-Verlag, New York, 1975.

[18] L. Grüne, *Numerical stabilization of bilinear control systems*, SIAM J. Control Optim., **34** (1996), pp. 2024-2050.

[19] L. Grüne, *An adaptive grid scheme for the discrete Hamilton-Jacobi-Bellman equation*, Numer. Math., **75** (1997), pp. 319–337.

[20] R. Gonzáles and C. Sagastizábal, *Un algorithme pour la résolution rapide d'équations discrètes de Hamilton-Jacobi-Bellman*, C.R. Acad. Sci. Paris, Sér. I, **311** (1990), pp. 45–50.

[21] R.A. Howard, Dynamic programming and Markov processes, Wiley, New York, 1960.

[22] R. Kalaba, *On nonlinear differential equations, the maximum operation and monotone convergence*, J. of Math. Mech., **8** (1959), pp. 519–574.
[23] K. Kunisch, S. Volkwein and L. Xie, *HJB-POD Based Feedback Design for the Optimal Control of Evolution Problems*, SIAM J. on Appl. Dyn. Syst., **4** (2004), pp. 701-722.
[24] K. Kunisch and L. Xie, *POD-Based Feedback Control of Burgers Equation by Solving the Evolutionary HJB Equation*, Comput. Math. Appl., **49** (2005), pp. 1113–1126.
[25] M. Pollatschek and B. Avi-Itzhak, Algorithms for Stochastic Games with Geometrical Interpretation, Management Sci., **15** (1969), pp. 399–415.
[26] M.L. Puterman and S.L. Brumelle, *On the convergence of Policy iteration in stationary Dynamic Programming.* Math. Oper. Res., **4** (1979), pp. 60–69.
[27] M.S. Santos, *Numerical solution of dynamic economic models*, in Handbook of Macroeconomics, J.B. Taylor abd M. Woodford eds., Elsevier Science, Amterdam, The Netherlands, pp. 311-386.
[28] A. Seeck, *Iterative lösungen der Hamilton-Jacobi-Bellman-Gleichung bei unendlichen Zeithorizont*, Diplomarbeit, Universität Kiel, 1997.
[29] M.S. Santos and J. Rust, *Convergence properties of policy iteration*, SIAM J. Control Optim., **42** (2004), pp. 2094–2115.
[30] J. A. Sethian, *Level set methods and fast marching methods*, Cambridge University Press, 1999.
[31] J. A. Sethian and A. Vladimirsky, *Ordered upwind methods for static Hamilton-Jacobi equations: theory and algorithms*, SIAM J. Numer. Anal., **41** (2003), pp. 325–363.
[32] M.M. Tidball and R. González, *Fast solution of general nonlinear fixed point problems*, in "System modelling and optimization (Zurich, 1991)", Lecture Notes in Control and Inform. Sci., 180, Springer, Berlin, 1992, 35–44.
[33] F. Tröltzsch, Optimal Control of Partial Differential Equations: Theory, Methods and applications, AMS 2010.
[34] S. Volkwein, *Model Reduction using Proper Orthogonal Decomposition*, Lecture notes, 2011 `www.math.uni-konstanz.de/numerik/personen/volkwein/teaching/POD-Vorlesung.pdf`