monviso: A Python Package for Solving Monotone Variational Inequalities

Nicola Mignoni, Reza Rahimi Baghbadorani, Raffaele Carli,
Peyman Mohajerin Esfahani, Mariagrazia Dotoli, and Sergio Grammatico

***Abstract*— In this paper, we present monviso (monotone variational inequalities solver), a novel open-source Python package for solving monotone variational inequalities. We detail the package's structure and baseline functionality, discussing a simple example that illustrates the essential methods and parameters. Moreover, we characterize how the proximal operator, which is the foundation of many iterative schemes, is handled through cvxpy, an open-source Python library for convex optimization. We list the available algorithms and describe the basic implementation of any general iterative method to enable users to build additional and (possibly new) algorithms. Finally, we illustrate several examples of possible use cases for monviso, showcasing the different applications the package can support across various fields, including control, optimization, dynamic game theory, and machine learning.**

I. Introduction

The proliferation of open-source projects that allow to easily implement control and decision-making problems is positively impacting both the academic and industrial communities. On the one hand, it allows researchers to accelerate the pace of research by providing unified frameworks for testing and benchmarking new ideas and approaches. On the other hand, it enables practitioners to solve real-world problems, often by means of quick prototyping before devising *ad-hoc* tailored implementations. Such projects serve as essential resources to democratize access to often advanced computational tools, removing the financial burden associated with proprietary software and allowing for customization and extensions aimed at satisfying specific needs. Moreover, they allow users even from outside the control and optimization community to tackle their own problems without the need of delving into technical details. Among the available mathematical tools, variational inequalities (VIs) unify many of the problems that appear in control and decision science at large, encompassing, e.g., optimal control [1], [2], optimization [3], [4], machine learning [5], game theory [6], [7], [8], and finance [9], making it a deeply studied topic of research [10]. Although the majority of existing projects strongly focus on optimization [11], and, to a lesser extent, control and dynamical systems [12], VIs have received limited attention from the perspective of systematic numerical implementations. Nevertheless, the research literature on iterative methods for solving monotone VIs keeps growing consistently. To the best of the authors' knowledge, only one package for solving VI targets Python (`imgemp/VI-Solver`), while two packages for Julia are dedicated to finite-dimensional VIs (`VariationalInequality.jl`) and complementarity problems (`Complementarity.jl`). A dedicated suite is provided by GAMS; albeit being closed source, this encompasses tools for modelling and solving VIs, quasi-VIs, and equilibrium problems. Note that we are here focusing on *modelling protocols*, i.e., the package/language that allows for declaring the problem at hand, and not *solvers*, i.e., the software that computes its solution. Except for GAMS, the surveyed pieces of software have not yet reached a mature development stage. Moreover, the number of methods and algorithms they implement for VI resolution is limited.

In this paper, we provide an overview of `monviso` (`mon`otone `v`ariational `i`nequalities `so`lver), a novel Python package for solving monotone VIs. Although programming languages such as MATLAB and Julia are specifically tailored to numerical programming, Python has been chosen as the base language for `monviso` due to its ever-increasing popularity and ease of use. Essentially, `monviso` acts as a wrapper around `cvxpy` [13], one of the most used pieces of software for convex optimization, in order to simplify the process of implementing montone VIs. Differently from the aforementioned projects, `monviso` has been designed to be modular and simple to extend: the algorithms implemented in the current version, in fact, do not include all the ones existing in literature. Therefore, allowing for simply adding (possibly new) iterative methods is crucial for the package's fruibility. `monviso`, together with all the code snippets discussed in the following, is publicly available at

`github.com/nicomignoni/monviso`

Moreover, all details here reported refer to the current package version, i.e., `0.1`.

The work of N. Mignoni was supported by the Italian Ministry of University and Research under the National Operating Programme for Research and Innovation 2014-2020, Action IV.5, Decree no. 1061/2021. The work of R. Rahimi Baghbadorani, P. Mohajerin Esfahani, and S. Grammatico was supported by the ERC under the research project TRUST-91562 and COSMOS-91409. The work of R. Carli and M. Dotoli was supported by the NRRP - Mission 4 Component 2 Investment 1.3 - Call for tender No. 341 of March 15, 2022 of MUR - project "NEST (Network 4 Energy Sustainable Transition)" (project no. PE00000021) and NRRP - Mission 4 Component 2 Investment 1.1 - Call for "Projects of significant national interest - Progetti di rilevante interesse nazionale (PRIN)" - Decree no. 104 of February 2, 2022 of MUR (project: CORRECT, code: 202248PWRY).

N. Mignoni, R. Carli, and M. Dotoli are with the Department of Electrical and Information Engineering of the Polytechnic of Bari, Italy (e-mail: {nicola.mignoni, raffaele.carli, mariagrazia.dotoli}@poliba.it). R. Rahimi-Bagbadorani, P. Mohajerin Esfahani, and S. Grammatico are with the Delft Center for Systems and Control of TU Delft, The Netherlands (e-mail: {r.rahimibaghbadorani, p.mohajerinesfahani, s.grammatico}@tudelft.ne).

II. Notation and Preliminaries

In the sequel, set $\mathbb{R}$ and $\mathbb{N}$ represent the real and natural numbers, respectively. We denote $[\mathbf{x}_1^\top, \dots, \mathbf{x}_N^\top]^\top =: \mathrm{col}(\mathbf{x}_i)_{i=1}^M$ and $[\mathbf{x}_1, \dots, \mathbf{x}_N] =: \mathrm{row}(\mathbf{x}_i)_{i=1}^M$. The all-zeros and all-ones vectors are denoted with $\mathbf{1}$ and $\mathbf{0}$, respectively; the identity matrix of size $N \times N$ is $\mathbf{I}_N$, while vector $\mathbf{e}_{i,N}$ denotes the i-th row of $\mathbf{I}_N$. Given a square matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, its positive (negative) semidefiniteness is denoted with $\mathbf{A} \succeq 0$ ($\mathbf{A} \preceq 0$). The diagonal block matrix made by matrices $\mathbf{A}_1, \dots, \mathbf{A}_n$ is denoted as $\mathrm{blkdiag}(\mathbf{A}_1, \dots, \mathbf{A}_n)$. Symbols $\otimes$ and $\odot$ denote the Kronecker and Hadamard (i.e., element-wise) product, respectively. The golden ratio constant is denoted as $\varphi := \frac{1+\sqrt{5}}{2}$. The uniform distribution bounded between $a, b \in \mathbb{R}$ is denoted as $\mathcal{U}(a, b)$. Given a set $\mathcal{S} \subseteq \mathbb{R}^n$, the associated indicator function is denoted as $\iota_{\mathcal{S}} : \mathbb{R} \to \{0, +\infty\}$, so that $\iota_{\mathcal{S}}(\mathbf{x}) = 0$ if $\mathbf{x} \in \mathcal{S}$ and $\iota_{\mathcal{S}}(\mathbf{x}) = +\infty$ otherwise. The normal cone associated with set $\mathcal{S}$ is $\mathsf{N}_{\mathcal{S}} : \mathcal{S} \rightrightarrows \mathbb{R}^n$, defined as $\mathsf{N}_{\mathcal{S}}(\mathbf{x}) := \{\mathbf{y} \in \mathbb{R}^n : \mathbf{y}^\top(\mathbf{x} - \mathbf{z}) \geq 0,\ \forall \mathbf{z} \in \mathcal{S}\}$. A mapping $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^n$ is L-Lipschitz iff $\|\mathbf{F}(\mathbf{x}) - \mathbf{F}(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\|$, for an arbitrary $L < +\infty$. Moreover, a mapping $\mathbf{F}(\cdot)$ is *monotone* if $(\mathbf{F}(\mathbf{x}) - \mathbf{F}(\mathbf{y}))^\top(\mathbf{x} - \mathbf{y}) \geq 0$ holds for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, and *strongly monotone* if $(\mathbf{F}(\mathbf{x}) - \mathbf{F}(\mathbf{y}))^\top(\mathbf{x} - \mathbf{y}) \geq \mu\|\mathbf{x} - \mathbf{y}\|^2$ holds for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and some $\mu > 0$. The constrained proximal operator for a given point $\mathbf{x} \in \mathbb{R}^n$ and a scalar function $g : \mathbb{R}^n \to \mathbb{R}$, with respect to set $\mathcal{S}$, is defined as

$$\mathrm{prox}_{g,\mathcal{S}}(\mathbf{x}) = \operatorname*{arg\,min}_{\mathbf{y}\in\mathcal{S}} \left\{ g(\mathbf{y}) + \frac{1}{2}\|\mathbf{y} - \mathbf{x}\|^2 \right\}. \quad (1)$$

The projection operator for point $\mathbf{x} \in \mathbb{R}^n$ with respect to set $\mathcal{S}$ can be defined as (1) when $g(\mathbf{y}) = 0$, i.e.,

$$\mathrm{proj}_{\mathcal{S}}(\mathbf{x}) = \mathrm{prox}_{0,\mathcal{S}}(\mathbf{x}) = \operatorname*{arg\,min}_{\mathbf{y}\in\mathcal{S}} \frac{1}{2}\|\mathbf{y} - \mathbf{x}\|^2. \quad (2)$$

Given a vector mapping $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^n$ and a scalar convex (possibly non-smooth) function $g : \mathbb{R}^n \to \mathbb{R}$, solving a VI consists of finding a point $\mathbf{x}^* \in \mathbb{R}^n$ such that the following holds:

$$\inf_{\mathbf{x}\in\mathbb{R}^n} (\mathbf{x} - \mathbf{x}^*)^\top \mathbf{F}(\mathbf{x}^*) - g(\mathbf{x}) - g(\mathbf{x}^*) \geq 0. \quad (3)$$

The standing assumptions for (3) are (i) the *monotonicity* of $\mathbf{F}(\cdot)$ always holds; (ii) $g(\cdot)$ being proper convex lower-semicontinuous; the iii) the non-emptiness of the solution set of (3). When a VI is constrained to a set $\mathcal{S} \subset \mathbb{R}^n$, by letting $g(\mathbf{x}) = \iota_{\mathcal{S}}(\mathbf{x})$, the problem in (3) becomes

$$\inf_{\mathbf{x}\in\mathcal{S}} (\mathbf{x} - \mathbf{x}^*)^\top \mathbf{F}(\mathbf{x}^*) \geq 0. \quad (4)$$

III. The Package Overview

`monviso` is a novel open-source Python package for solving monotone VIs. The proposed package is a wrapper around `cvxpy` [13], which allows to conveniently define the proximal and projection operators. These operators, in fact, usually represent the most computationally intense steps of iterative schemes, being themselves fully-fledged optimization problems. Albeit some proximal operators can be explicitly expressed in analytical form, the majority requires a (usually convex) solver for retrieving the solution. A list of *proximal-friendly* operators is reported in [14], specifically cataloging $g(\cdot)$ functions in (3), which yields an exact formulation for the proximity operator. Nonetheless, evaluating its analytic form, even when possible, is usually tedious and error-prone, binding the resulting iterative approach to the specific application at hand. Therefore, `monviso` follows the same philosophy of several existing optimization modelling language, which allow for a degree of expressiveness close to mathematical statements. This enables the user to input a generic form for g and $\mathcal{S}$, which will be used to automatically construct (1).

A. The Basic Functionality

In order to illustrate the inner working of `monviso`, let us provide a simple example as a starting point.

Example 1: Let $\mathbf{F}(\mathbf{x}) = \mathbf{H}\mathbf{x}$ for some $\mathbf{H} \succ 0$, $g(\mathbf{x}) = \|\mathbf{x}\|_1$, and $\mathcal{S} = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$, for some $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{b} \in \mathbb{R}^n$. It is straightforward to verify that $\mathbf{F}(\cdot)$ is strongly monotone with $\mu = \lambda_{\min}(\mathbf{H})$ and Lipschitz with $L = \|\mathbf{H}\|_2$. Therefore, the resulting VI in (3) could be solved using the well-known proximal gradient descent method [NEM83], where convergence is guaranteed for a step size $\lambda \in \left(0, \frac{2\mu}{L^2}\right)$. The following Python code illustrates how to solve the VI in (3) using `monviso`.

```
1  import numpy as np
2  import cvxpy as cp
3
4  from monviso import VI
5
6  # Create the problem data
7  n, m = 30, 40
8  H = np.random.uniform(2, 10, size=(n, n))
9  A = np.random.uniform(45, 50, size=(m, n))
10 b = np.random.uniform(3, 7, size=(m,))
11
12 # Make H positive definite
13 H = H @ H.T
14
15 # Lipschitz and strong monotonicity constants
16 mu = np.linalg.eigvals(H).min()
17 L = np.linalg.norm(H, 2)
18
19 # Define F, g, and S
20 F = lambda x: H @ x
21 g = lambda x: cp.norm(x)
22 S = [lambda x: A @ x <= b]
23
24 # Define and solve the VI
25 vi = VI(n, F, g, S)
26
27 x0 = np.random.uniform(4, 5, n)
28 algorithm_params = {
29     "x": x0,
30     "step_size": 2 * mu / L**2
31 }
32 sol = vi.solution(
33     "pg", algorithm_params, max_iters=25,
34     eval_tol=-np.inf, log_path="result.log"
35 )
```

Except for lines 1-25, which simply define the problem parameters and needed constants, the core implementation of the VI problem is expressed by lines 27-37. In `monviso`, a VI is an instance of the class `VI()`, which takes the following objects as attributes:

- `n`: the size of the vector space n. This is necessary since `monviso` cannot infer the size of the vector space solely from `F` or `g`.
- `F`: any Python function[1] taking a `numpy` array as input, and returning another `numpy` array of the same size. It corresponds to the operator $\mathbf{F}(\cdot)$.
- `g`: a function having a single `cvxpy Variable` as argument and returning a scalar. Moreover, such a variable must be a 1-dimension array, i.e., a vector. This characterization corresponds to the mathematical definition of $g(\cdot)$ in (3). As default, `g = 0`.
- `S`: a Python `list` of callables returning a `cvxpy`'s `Constraints`. Each of them has a single `cvxpy`'s `Variable` as variable. As for `g`, such a variable must be a 1-dimension array, i.e., a vector. As default, `S = []`.

The rationale for requiring the vector space size and the variable characterizing `g` and `S` is detailed in the following. In case `g` and `S` are provided, `monviso` will check whether they are characterized by a single and identical variable vector, in order to ensure consistency with the VI definition in (3). An initial solution is randomly generated in line 30, while the parameters characterizing the employed algorithm are declared in line 31. The parameters to be declared depend on the chosen iterative method, although they generally comprise (at least) one starting solution and the (possibly initial) value of λ, referred to as `step_size`. The VI solution is evaluated in line 34 through the `vi.solution()` function, which takes as mandatory arguments: i) the name of the algorithm to use, ii) its parameters (i.e., the key-value map previously discussed), and iii) the maximum number of iterations. Additional optional parameters are also available:

- `eval_func`: unless the maximum number of allowed iterations is reached, iterative algorithm stop when a certain (arbitrarily small) small tolerance is reached. `monviso` allows for defining custom evaluation functions, which take the k-th iteration solution, $\mathbf{x}_k$, as single argument. As default, the *residual* magnitude $J : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ is set, defined as $J(\mathbf{x}_k) := \|\mathbf{x}_k - \text{prox}_{g,\mathcal{S}}(\mathbf{x}_k - \mathbf{F}(\mathbf{x}_k))\|$
- `eval_tol`: the tolerance value for the evaluation function, under which the chosen iterative algorithm stops. As tolerance value of 1^{-9} is set as the default.
- `log_path`: the path where the algorithm log is saved. Such a log is a comma separated value file, comprising the following fields: i) `iter`, i.e., the iteration number, ii) `eval_func_value`, i.e., the value of the evaluation function, and iii) `time`, i.e., the time (in seconds) elapsed from the first to the k-th iteration.

B. Stateful Constrained Proximal Operator

As per its definition in (1), the evaluation of the proximal operator corresponds to finding its minimizer. In `monviso`, this is accomplished by defining (1) as a `cvxpy.Problem`. Usually, optimization modelling languages that allow the user to define the problem declaratively need to parse it in order to provide the solver at hand with the accepted canonical form. This is often done through an epigraphic reformulation, which casts the constraint to a suitable conic representation. In our use case, the challenge arises because this parsing process would occur at each iteration, possibly multiple times when the proximal operator needs to be evaluated more than once (e.g., extragradient method [KOR76]). Such a repeated parsing process would dramatically worsen the overall computation time.

In order to mitigate this potential overhead, we exploited the disciplined parametrized programming (DPP) that `cvxpy` implements. DPP defines a ruleset for parsing an optimization problem in canonical form while keeping track of constant terms that might be changed from one solution instance to another. A detailed discussion of the DPP principles can be found in [15]. For our case, the term $\mathbf{x}$ is a parameter for (1) and (2): on such a basis, `monviso` implements the proximal and projection[2] operators as invariant objects upon instantiating the `VI()`. Their *statefulness* resides in the fact that the same instance is used in all the iterations for any given algorithm, with only $\mathbf{x}$ (i.e., the *state*) being updated.

Moreover, the definition of proximal operator in (1) includes a constraint set, differing from the usual definition where the minimization problem is defined over $\mathbb{R}^n$. The reason lies in the difficulties behind the computational instabilities that arise from implementing the indicator function. In fact, since $g(\cdot)$ might be non-smooth, one can set $g(\mathbf{x}) = \iota_{\mathcal{S}}(\mathbf{x})$ when the VI at hand is constrained by $\mathcal{S}$. However, gradient-based methods are known to be dramatically susceptible to the behavior of strongly discontinuous functions [16]. Characterizing the proximal operator as constrained counteracts such issues, with `S = []` corresponding to $\mathcal{S} = \mathbb{R}^n$.

C. The Implemented Algorithms

Table I lists the algorithms implemented in the current version of `monviso`, along with each method's name and the required parameters. For the sake of brevity, we will not delve into the details and properties of each algorithm: the package documentation reports a brief description of each method, together with the description of the basic iterations and its convergence conditions.

Each implemented algorithm shares the same function signature, which can be summarized as follows:

```
1    def algorithm_name(
2        x: np.ndarray,
3        step_size: float,
4        **other_params: dict, optional,
5        **cvxpy_solve_params: dict, optional
6    ) -> np.ndarray:
7        # (Eventual) preparatory steps
8        while True:
9            # Iteration steps, out of which at
10           # least one generates x
11           yield x
```

[1] In Example 1, an anonymous function.

[2] Albeit (2) is a special case of (1), it is convenient to instantiate it as standalone operator, since it is often used for evaluations not (always) strictly related to algorithms iterates, e.g., the residual metric used for stopping the convergence.

TABLE I

THE LIST OF IMPLEMENTED ALGORITHMS

Ref.	Method	Algorithm	Parameters
[NEM83]	`pg`	Proximal Gradient	$\mathbf{x}_0 \in \mathbb{R}^n$, $\lambda \in \left(0, \frac{1}{L}\right)$
[KOR76]	`eg`	Extragradient	$\mathbf{x}_0 \in \mathbb{R}^n$, $\lambda \in \left(0, \frac{1}{L}\right)$
[POP80]	`popov`	Popov's Method	$\mathbf{x}_0, \mathbf{y}_0 \in \mathbb{R}^n$, $\lambda \in \left(0, \frac{1}{2L}\right)$
[TSE00]	`fbf`	Forward-Backward-Forward	$\mathbf{x}_0 \in \mathbb{R}^n$, $\lambda \in \left(0, \frac{1}{L}\right)$
[MAL20a]	`frb`	Forward-Reflected-Backward	$\mathbf{x}_0, \mathbf{x}_1 \in \mathbb{R}^n$, $\lambda \in \left(0, \frac{1}{2L}\right)$
[MAL15]	`prg`	Projected Reflected Gradient	$\mathbf{x}_0, \mathbf{x}_1 \in \mathbb{R}^n$, $\lambda \in \left(0, \frac{\sqrt{2}-1}{L}\right)$
[YOO21]	`eag`	Extra Anchored Gradient	$\mathbf{x}_0 \in \mathbb{R}^n$, $\lambda \in \left(0, \frac{1}{\sqrt{3}L}\right)$
[CAI20]	`arg`	Accelerated Reflected Gradient	$\mathbf{x}_0, \mathbf{x}_1 \in \mathbb{R}^n$, $\lambda \in \left(0, \frac{1}{12L}\right)$
[YOO21]	`fogda`	(Explicit) Fast OGDA	$\mathbf{x}_0, \mathbf{x}_1, \mathbf{y} \in \mathbb{R}^n$ $\lambda \in \left(0, \frac{1}{4L}\right)$, $\alpha > 2$
[SED23]	`cfogda`	Constrained Fast OGDA	$\mathbf{x}_0, \mathbf{y}_0 \in \mathbb{R}^n$, $\mathbf{x}_1 \in \mathcal{S}$ $\mathbf{z}_0 \in \mathsf{N}_{\mathcal{S}}(\mathbf{x}_1)$, $\lambda \in \left(0, \frac{1}{4L}\right)$, $\alpha > 2$
[MAL20b]	`graal`	Golden Ratio Algorithm	$\mathbf{x}_0, \mathbf{x}_1 \in \mathbb{R}^n$, $\lambda \in \left(0, \frac{\varphi}{2L}\right)$, $\phi \in (1, \varphi]$
[MAL20b]	`agraal`	Adaptive Golden Ratio Algorithm	$\mathbf{x}_0, \mathbf{x}_1 \in \mathbb{R}^n$, $\lambda_0 > 0$, $\phi \in (1, \varphi]$
[BAG24]	`hgraal_1`	Hybrid Golden Ratio Algorithm 1	$\mathbf{x}_0, \mathbf{x}_1 \in \mathbb{R}^n$, $\lambda_0 > 0$, $\phi \in (1, \varphi]$
[BAG24]	`hgraal_2`	Hybrid Golden Ratio Algorithm 2	$\mathbf{x}_0, \mathbf{x}_1 \in \mathbb{R}^n$, $\lambda_0 > 0$, $\phi \in (1, \varphi]$

Each iterative algorithm is, thus, implemented as a generator, indefinitely yielding the new value of $\mathbf{x}$ at each function call. As mentioned in Example 1, each algorithm takes as a mandatory argument at least one starting point `x`, as well as the (possibly initial) value for the `step_size`. More arguments can be declared: for instance, both the Explicit and Constrained Fast Optimistic Gradient Descent-Ascent (OGDA) require a further parameter α to be set. Wherever possible, we set additional arguments with a suitable default in order to simplify the usage. The additional set of optional parameters defined by `cvxpy_solve_params` contains the arguments that can be passed to the `cxvpy.Problem.solve()` function. Specifically, since (1) is implemented as `cvxpy` optimization problem, as mentioned in Section III-B, the user might need to pass further arguments to explicitly set `cvxpy` options on how to solve such a problem (e.g., which solver to use or parsing options). The algorithms' function signature strives to allow users to possibly implement their own iterative methods quickly and easily. In fact, one of the core concepts behind `monviso` is keeping modularity and ease of use as the package might grow in the future, anticipating future growth with the addition of more algorithms for monotone VIs.

IV. APPLICATION EXAMPLES

In this section, we illustrate some examples related to control, optimization, game theory, and machine learning problems, which can be reduced to a VI and solved through `monviso`. For the sake of brevity, we provide a short description of the proposed examples here; further details can be found in the available repository[3].

1) Linear Complementarity Problem [4, Section 3]: A common problem that can be cast to a VI is the linear complementarity problem: given $\mathbf{b} \in \mathbb{R}^n$ and $0 \preceq \mathbf{A} \in \mathbb{R}^{n \times n}$, one want to solve the following

$$\text{find } \mathbf{x} \in \mathbb{R}^n_{\geq 0} \text{ s.t. } \mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{b} \geq \mathbf{0},\ \mathbf{y}^\top \mathbf{x} = 0. \quad (5)$$

By setting $\mathbf{F}(\mathbf{x}) = -\mathbf{y} = -\mathbf{A}\mathbf{x} - \mathbf{b}$ and $\mathcal{S} = \mathbb{R}_{\geq 0}$ it can be readily verified that each solution for (4) is also a solution for (5). Figure 1a illustrates the convergence results.

2) Two Players Zero-Sum Game [17]: Many examples of non-cooperative behavior between two adversarial agents can be modelled through zero-sum games. Let us consider vectors $\mathbf{x}_i \in \Delta_i$ as the decision variable of the i-th player, with $i \in \{1, 2\}$, where $\Delta_i \subset \mathbb{R}^{n_i}$ is the simplex constraints set defined as $\Delta_i := \{\mathbf{x} \in \mathbb{R}^{n_i} : \mathbf{1}^\top \mathbf{x} = 1\}$, for all $i \in \{1, 2\}$. Let $\mathbf{x} := \text{col}(\mathbf{x}_i)_{i=1}^2$. Players try to solve the following problem:

$$\min_{\mathbf{x}_1 \in \Delta_1} \max_{\mathbf{x}_2 \in \Delta_2} \Phi(\mathbf{x}_1, \mathbf{x}_2) \quad (6)$$

whose (Nash) equilibrium solution is achieved for $\mathbf{x}^*$ satisfying the following

$$\Phi(\mathbf{x}_1^*, \mathbf{x}_2) \leq \Phi(\mathbf{x}_1^*, \mathbf{x}_2^*) \leq \Phi(\mathbf{x}_1, \mathbf{x}_2^*), \quad \forall \mathbf{x} \in \Delta_1 \times \Delta_2 \quad (7)$$

For the sake of simplicity, we consider $\Phi(\mathbf{x}_1, \mathbf{x}_2) := \mathbf{x}_1^\top \mathbf{H} \mathbf{x}_2$, for some $\mathbf{H} \in \mathbb{R}^{n_1 \times n_2}$. In doing so, the equilibrium condition in the previous equation can be written as a VI, where $\mathcal{S} = \Delta_1 \times \Delta_2$ and the mapping $\mathbf{F} : \mathbb{R}^{n_1 + n_2} \to \mathbb{R}^{n_1 + n_2}$ is defined:

$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} \mathbf{H}\mathbf{x}_1 \\ -\mathbf{H}^\top \mathbf{x}_2 \end{bmatrix} = \begin{bmatrix} & \mathbf{H} \\ -\mathbf{H}^\top & \end{bmatrix} \mathbf{x} \quad (8)$$

The convergence results are reported in Fig. 1b.

3) Feasibility Problem: Finding a point in the intersection of M balls [3]: Let us consider M balls in $\mathbb{R}^n$, where the i-th ball of radius $r_i > 0$ centered in $\mathbf{c}_i \in \mathbb{R}^n$ is given by $\mathcal{B}_i(\mathbf{c}_i, r_i) \subset \mathbb{R}^n$. We are interested in finding a point belonging to their intersection, i.e., we want to solve the following:

$$\text{find } \mathbf{x} \text{ subject to } \mathbf{x} \in \bigcap_{i=1}^{M} \mathcal{B}_i(\mathbf{c}_i, r_i) \quad (9)$$

[3] `github.com/nicomignoni/monviso/tree/master/examples`

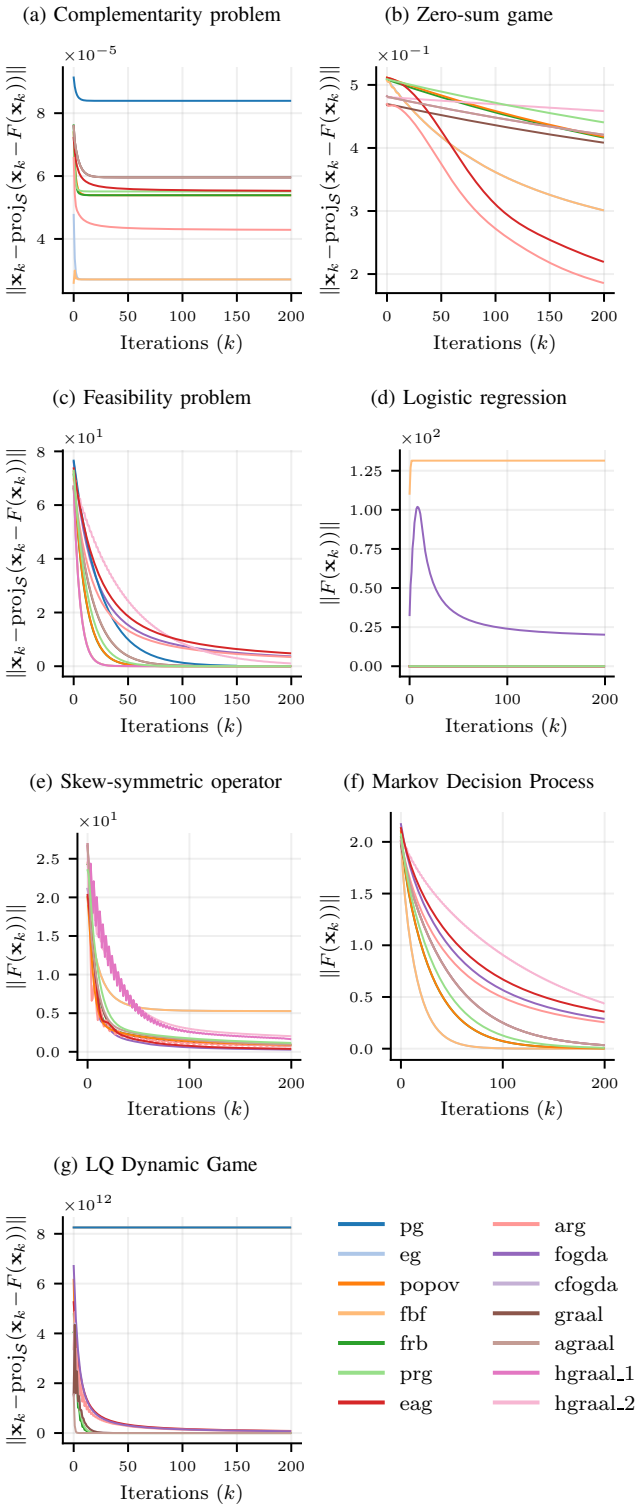

Fig. 1. Convergence results for each example, using the currently available algorithms in `monviso`.

It is straightforward to verify that the projection of a point onto $\mathcal{B}_i(\mathbf{c}_i, r_i)$ is evaluated as:

$$\mathrm{proj}_{\mathcal{B}_i(\mathbf{c}_i,r_i)}(\mathbf{x}) = \begin{cases} r_i \dfrac{\mathbf{x}-\mathbf{c}_i}{\|\mathbf{x}-\mathbf{c}_i\|} & \text{if } \|\mathbf{x}-\mathbf{c}_i\| > r_i \\ \mathbf{x} & \text{otherwise} \end{cases} \tag{10}$$

Due to the non-expansiveness of the projection in (2), one can find a solution for (1) as the fixed point of the following iterate:

$$\mathbf{x}_{k+1} = \mathsf{T}(\mathbf{x}_k) = \frac{1}{M}\sum_{i=1}^{M} \mathrm{proj}_{\mathcal{B}_i(\mathbf{c}_i,r_i)}(\mathbf{x}_k) \tag{11}$$

which result from the well-known Krasnosel'skiĭ-Mann iterate. By letting $F = \mathsf{I} - \mathsf{T}$, where I denotes the identity operator, the fixed point for (3) can be treated as the canonical VI. Figure 1c illustrates the convergence results.

4) Skew symmetric operator [18, Example 20.35]: A simple example of a monotone operator that is not (even locally) strongly monotone is the skewed-symmetric operator, which is described as $\mathbf{F}(\mathbf{x}) = \mathrm{blkdiag}(\mathbf{A}_1, \dots, \mathbf{A}_M)\mathbf{x}$ for some arbitrary $M \in \mathbb{N}$, where $\mathbf{A}_i = \mathrm{triu}(\mathbf{B}_i) - \mathrm{tril}(\mathbf{B}_i)$, for some $\mathbf{B}_i \succeq 0$, for all $i = 1, \dots, M$. The convergence results are reported in Fig. 1e.

5) Sparse logistic regression [5, Section 3]: Consider a dataset of M rows and N columns, so that $\mathbf{A} = \mathrm{col}(\mathbf{a}_i^\top)_{i=1}^M \in \mathbb{R}^{M\times N}$ is the dataset matrix, and $\mathbf{a}_i \in \mathbb{R}^N$ is the i-th features' vector for the i-th dataset row. Moreover, let $\mathbf{b} \in \mathbb{R}^M$ be the target vector, so that $b_i \in \{-1, 1\}$ is the (binary) ground truth for the i-th data entry. The sparse logistic regression consists of finding the weight vector $\mathbf{x} \in \mathbb{R}^N$ that minimizes the loss function $V : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$, defined as follows

$$\begin{aligned} V(\mathbf{x}) &:= \sum_{i=1}^{M} \log\left(1 + \frac{1}{\exp(b_i \mathbf{a}_i^\top \mathbf{x})}\right) + \gamma\|\mathbf{x}\|_1 \\ &= \underbrace{\mathbf{1}_M^\top \log(1 + \exp(-\mathbf{b} \odot \mathbf{A}\mathbf{x}))}_{=:s(\mathbf{x})} + \underbrace{\gamma\|\mathbf{x}\|_1}_{=:g(\mathbf{x})} \end{aligned} \tag{12}$$

where $\gamma \in \mathbb{R}_{>0}$ is the ℓ_1-regulation strength. The gradient for $s(\cdot)$, $\nabla s_{\mathbf{x}}(\mathbf{x})$, is calculated as

$$\nabla s_{\mathbf{x}}(\mathbf{x}) = -\frac{\mathbf{A}^\top \odot (\mathbf{1}_N \otimes \mathbf{b}^\top) \odot \exp(-\mathbf{b} \odot \mathbf{A}\mathbf{x})}{1 + \exp(-\mathbf{b} \odot \mathbf{A}\mathbf{x})} \mathbf{1}_M \tag{13}$$

The problem of finding the minimizer for (12) can be cast as (3), with $\mathbf{F}(\mathbf{x}) := \nabla s_{\mathbf{x}}(\mathbf{x})$. The convergence trajectory is reported in Fig. 1d.

6) Markov Decision Process (MDP) [19, Chapter 3]: A stationary discrete MDP is characterized by the tuple $(\mathcal{X}, \mathcal{A}, P, r, \gamma)$, i) where $\mathcal{X}$ is the (finite countable) set of states; ii) $\mathcal{A}$ is the (finite countable) set of actions; iii) $P : \mathcal{X} \times \mathcal{A} \times \mathcal{X} \to [0, 1]$ is the transition probability function, such that $P(x, a, x^+)$ is the probability of ending up in state $x^+ \in \mathcal{S}$ from state $x \in \mathcal{X}$ when taking action $a \in \mathcal{A}$; iv) $r : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is the reward function, so that $r(x, x^+)$ returns the reward for transitioning from state $x \in \mathcal{X}$ to state $x^+ \in \mathcal{X}$; $\gamma \in \mathbb{R}_{>0}$ is a discount factor. The aim is to find a policy, i.e., a function $\pi : \mathcal{S} \to \mathcal{A}$, returning the best action for any given state. A solution concept for MDP is the *value function*, $v^\pi : \mathcal{S} \to \mathbb{R}$, defined as

$$v^\pi(x) = \overbrace{\sum_{x^+ \in \mathcal{X}} P(x, \pi(x), x^+)\left(r(x, x^+) + \gamma v(x^+)\right)}^{=:\mathsf{T}(v^\pi)} \tag{14}$$

returning the "goodness" of policy π. The expression in (14) is known as *Bellman equation*, and can be expressed as an operator of v^π, i.e., $\mathsf{T}[v^\pi(s)] =: \mathsf{T}(v^\pi)$. It can be shown that the value function yielded by the optimal policy, v^*, results from the fixed-point problem $v^* = \mathsf{T}(v^*)$. Therefore, the latter can be formulated as a canonical VI, with $F = \mathsf{I} - \mathsf{T}$. The convergence results are reported in Fig. 1f.

7) Linear-Quadratic (LQ) Dynamic Games [1]: As shown in [1, Proposition 2], the receding horizon open-loop Nash equilibria (NE) can be reformulated as a non-symmetric VI. Specifically, consider a set of agents $\mathcal{N} = \{1, \dots, N\}$ characterizing a state vector $\mathbf{x}[t] \in \mathbb{R}^n$, whose (linear) dynamics is described as

$$\mathbf{x}[t+1] = \mathbf{A}\mathbf{x}[t] + \sum_{i \in \mathcal{N}} \mathbf{B}_i \mathbf{u}_i[t] \tag{15}$$

for $t = 1, \dots, T$. Each agent i selfishly tries to choose $\mathbf{u}_i[t] \in \mathbb{R}^m$ in order to minimize the following cost function

$$J_i(\mathbf{u}_i|\mathbf{x}_0, \mathbf{u}_{-i}) = \frac{1}{2} \sum_{t=0}^{T-1} \|\mathbf{x}[t|\mathbf{x}_0, \mathbf{u}]\|^2_{\mathbf{Q}_i} + \|\mathbf{u}_i[t]\|^2_{\mathbf{R}_i} \tag{16}$$

for some $0 \preceq \mathbf{Q}_i \in \mathbb{R}^{n \times n}$ and $0 \prec \mathbf{R}_i \in \mathbb{R}^{m \times m}$, with $\mathbf{u}_{-i} = \mathrm{col}(\mathbf{u}_j)_{j \in \mathcal{N} \setminus \{i\}}$ and $\mathbf{u}_j = \mathrm{col}(\mathbf{u}_j[t])_{t=1}^T$. Moreover, $\mathbf{u} = \mathrm{col}(\mathbf{u}_i)_{i \in \mathcal{N}}$. The set of feasible inputs, for each agent $i \in \mathcal{N}$, is $\mathcal{U}_i(\mathbf{x}_0, \mathbf{u}_{-i}) := \{\mathbf{u}_i \in \mathbb{R}^{mT} : \mathbf{u}_i[t] \in \mathcal{U}_i(\mathbf{u}_{-i}[t]),\ \forall t = 0, \dots, T-1;\ \mathbf{x}[t|\mathbf{x}_0, \mathbf{u}] \in \mathcal{X},\ \forall t = 1, \dots, T\}$, where $\mathcal{X} \in \mathbb{R}^n$ is the set of feasible system states. Finally, $\mathcal{U}(\mathbf{x}_0) = \{\mathbf{u} \in \mathbb{R}^{mTN} : \mathbf{u}_i \in \mathcal{U}(\mathbf{x}_0, \mathbf{u}_{-i}),\ \forall i \in \mathcal{N}\}$. Following [1, Definition 1], the sequence of input $\mathbf{u}_i^* \in \mathcal{U}_i(\mathbf{x}_0, \mathbf{u}_{-i})$, for all $i \in \mathcal{N}$, characterizes an open-loop NE iff

$$J(\mathbf{u}_i^*|\mathbf{x}_0, \mathbf{u}_{-i}^*) \leq \inf_{\mathbf{u}_i \in \mathcal{U}_i(\mathbf{x}_0, \mathbf{u}_{-i}^*)} \{J(\mathbf{u}_i^*|\mathbf{x}_0, \mathbf{u}_{-i})\} \tag{17}$$

which is satisfied by the fixed-point of the best response mapping of each agent, defined as

$$\mathbf{u}_i^* = \underset{\mathbf{u}_i \in \mathcal{U}(\mathbf{x}_0, \mathbf{u}_{-i}^*)}{\arg\min}\ J_i(\mathbf{u}_i|\mathbf{x}_0, \mathbf{u}_{-i}^*), \quad \forall i \in \mathcal{N} \tag{18}$$

Proposition 2 in [1] states that any solution of (3) is a solution for (18) when $\mathcal{S} = \mathcal{U}(\mathbf{x}_0)$ and $F : \mathbb{R}^{mTN} \to \mathbb{R}^{mTN}$, defined as

$$\begin{aligned} \mathbf{F}(\mathbf{u}) = &\mathrm{col}(\mathbf{G}_i^\top \bar{\mathbf{Q}}_i)_{i \in \mathcal{N}}(\mathrm{row}(\mathbf{G}_i)_{i \in \mathcal{N}}\mathbf{u} + \mathbf{H}\mathbf{x}_0) + \\ &\mathrm{blkdiag}(\mathbf{I}_T \otimes \mathbf{R}_i)_{i \in \mathcal{N}}\mathbf{u} \end{aligned} \tag{19}$$

where, for all $i \in \mathcal{N}$, $\bar{\mathbf{Q}}_i = \mathrm{blkdiag}(\mathbf{I}_{T-1} \otimes \mathbf{Q}_i, \mathbf{P}_i)$, $\mathbf{G}_i = \mathbf{e}_{1,T}^\top \otimes \mathrm{col}(\mathbf{A}_i^t \mathbf{B}_i)_{t=0}^{T-1} + \mathbf{I}_T \otimes \mathbf{B}_i$ and $\mathbf{H} = \mathrm{col}(\mathbf{A}^t)_{t=1}^T$. Matrix $\mathbf{P}_i$ results from the open-loop NE feedback synthesis as discussed in [1, Equation 6].

V. Conclusions

In this paper, we have presented `monviso`, a novel open-source Python package for solving monotone variational inequalities (VIs). We have detailed the basic functionalities of the packages through a simple introductory example and discussed how `monviso` integrates `cvxpy` to implement the proximal operator in a stateful fashion. Several examples presented the implemented algorithms, ranging from control and optimization to machine learning, showcasing `monviso` utility across different applications.

Future work will focus on improving the overall performance, possibly resorting to just-in-time compilation, as well as extending `monviso` to other commonly used programming languages, such as MATLAB and Julia.

References

[1] E. Benenati and S. Grammatico, "Linear-quadratic dynamic games as receding-horizon variational inequalities," *arXiv preprint arXiv:2408.15703*, 2024.

[2] R. R. Baghbadorani, E. Benenati, and S. Grammatico, "A douglas-rachford splitting method for solving monotone variational inequalities in linear-quadratic dynamic games," 2025.

[3] H. H. Bauschke and J. M. Borwein, "On projection algorithms for solving convex feasibility problems," *SIAM review*, vol. 38, no. 3, pp. 367–426, 1996.

[4] P. T. Harker and J. Pang, "For the linear complementarity problem," *Lectures in Applied Mathematics*, vol. 26, pp. 265–284, 1990.

[5] K. Mishchenko, "Regularized newton method with global convergence," *SIAM Journal on Optimization*, vol. 33, no. 3, pp. 1440–1462, 2023.

[6] P. Scarabaggio, R. Carli, S. Grammatico, and M. Dotoli, "Local generalized nash equilibria with nonconvex coupling constraints," *IEEE Transactions on Automatic Control*, 2024.

[7] N. Mignoni, J. Martinez-Piazuelo, R. Carli, C. Ocampo-Martinez, N. Quijano, and M. Dotoli, "A game-theoretical control framework for transactive energy trading in energy communities," in *2024 European Control Conference (ECC)*. IEEE, 2024, pp. 786–791.

[8] N. Mignoni, R. Carli, and M. Dotoli, "A noncooperative stochastic rolling horizon control framework for v1g and v2b scheduling in energy communities," in *2023 European Control Conference (ECC)*. IEEE, 2023, pp. 1–6.

[9] G. Pantazis, R. Rahimi Baghbadorani, and S. Grammatico, "Nash equilibrium seeking for a class of quadratic-bilinear Wasserstein distributionally robust games." `https://pure.tudelft.nl/ws/portalfiles/portal/226975637/JOTA_DR_games_final_PRG.pdf`, 2024.

[10] F. Facchinei, "Finite-dimensional variational inequalities and complementarity problems," 2003.

[11] R. Anand, D. Aggarwal, and V. Kumar, "A comparative analysis of optimization solvers," *Journal of Statistics and Management Systems*, vol. 20, no. 4, pp. 623–635, 2017.

[12] G. Peev, N. Simidjievski, and S. Džeroski, "Modeling of dynamical systems: a survey of tools and a case study," in *20th International Multiconference Information Society-IS*, 2017, pp. 15–18.

[13] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[14] G. Chierchia, E. Chouzenoux, P. L. Combettes, and J.-C. Pesquet, "The proximity operator repository," *User's guide http://proximity-operator. net/download/guide. pdf. Accessed*, vol. 6, 2020.

[15] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter, "Differentiable convex optimization layers," *Advances in neural information processing systems*, vol. 32, 2019.

[16] D. Jeong, S. Ham, J. Yang, Y. Hwang, S. Kwak, H. Hua, X. Xin, and J. Kim, "Numerical study of an indicator function for front-tracking methods," *Mathematical Problems in Engineering*, vol. 2022, no. 1, p. 7381115, 2022.

[17] C. E. Lemke and J. T. Howson, Jr, "Equilibrium points of bimatrix games," *Journal of the Society for industrial and Applied Mathematics*, vol. 12, no. 2, pp. 413–423, 1964.

[18] H. H. Bauschke, P. L. Combettes, H. H. Bauschke, and P. L. Combettes, *Convex analysis and monotone operator theory in Hilbert spaces*. Springer, 2017.

[19] R. S. Sutton, "Reinforcement learning: An introduction," *A Bradford Book*, 2018.