Applied Soft Computing Journal 159 (2024) 111654

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Automatic design of interpretable control laws through parametrized Genetic Programming with adjoint state method gradient evaluation

Francesco Marchetti [a,*], Gloria Pietropolli [b], Federico Julian Camerota Verdù [b], Mauro Castelli [c], Edmondo Minisci [d]

[a] *Department of Guidance, Navigation and Control Systems, German Aerospace Centre (DLR), Institute of Space Systems, Robert-Hooke-Straße 7, 28359, Bremen, Germany*
[b] *Dipartimento di Matematica e Geoscienze, Università degli Studi di Trieste, H2bis Building, Via Alfonso Valerio 12/1, Trieste, 34127, TS, Italy*
[c] *NOVA Information Management School (NOVA IMS), Universidade NOVA de Lisboa, Campus de Campolide, Lisbon, 1070-312, Portugal*
[d] *Intelligent Computational Engineering Laboratory (ICE-Lab), Department of Mechanical and Aerospace Engineering, University of Strathclyde, 75 Montrose Street, Glasgow, G1 1XJ, Scotland, United Kingdom*

ARTICLE INFO

Keywords:
Genetic Programming
Gradient descent
Adjoint state method
Control

ABSTRACT

This work investigates the application of a Local Search (LS) enhanced Genetic Programming (GP) algorithm to the control scheme's design task. The combination of LS and GP aims to produce an interpretable control law as similar as possible to the optimal control scheme reference. Inclusive Genetic Programming (IGP), a GP heuristic capable of promoting and maintaining the population diversity, is chosen as the GP algorithm since it proved successful on the considered task. IGP is enhanced with the Operators Gradient Descent (OPGD) approach, which consists of embedding learnable parameters into the GP individuals. These parameters are optimized during and after the evolutionary process. Moreover, the OPGD approach is combined with the adjoint state method to evaluate the gradient of the objective function. The original OPGD was formulated by relying on the backpropagation technique for the gradient's evaluation, which is impractical in an optimization problem involving a dynamical system because of scalability and numerical errors. On the other hand, the adjoint method allows for overcoming this issue. Two experiments are formulated to test the proposed approach, named Operator Gradient Descent - Inclusive Genetic Programming (OPGD-IGP): the design of a Proportional–Derivative (PD) control law for a harmonic oscillator and the design of a Linear Quadratic Regulator (LQR) control law for an inverted pendulum on a cart. OPGD-IGP proved successful in both experiments, being capable of autonomously designing an interpretable control law similar to the optimal ones, both in terms of shape and control gains.

1. Introduction

Genetic Programming (GP) [1] is a powerful algorithm to evolve computer programs, represented as trees, by iteratively selecting, recombining, and mutating a population of candidate solutions. Thanks to this symbolic representation, GP generates solutions that, differently to the ones achieved with other artificial intelligence (AI) techniques, may be interpreted (i.e., when the GP trees present a limited number of nodes) by domain experts. Nevertheless, the search performed by GP operators (crossover and mutation) is solely syntactic. Thus, there is no explicit parameter optimization during the evolutionary process. This can lead to evident drawbacks, as pointed out by Castelli et al. [2]. For instance, let us consider the scenario where the evolutionary search led to an individual with the following syntax $K(x) = x + \sin(x)$, while the optimal solution is $K^*(x) = 3.3x + 1.003\sin(0.0001x)$. Since there is no explicit parameters optimization, the solution $K(x)$ might be easily lost during the selection phase, leading to a very inefficient process. Including a Local Search (LS) routine in traditional GP has proven to be an effective method to overcome this limitation [2–4]. The advantages of embedding a gradient-based approach as an LS method in the evolutionary GP flow have emerged clearly in tasks such as symbolic regression [5] and image classification [6].

The objective of this study is to demonstrate that this combination can also play a critical role in control applications, where GP offers a compelling option for generating comprehensible control laws [7,8], thus providing a major benefit with respect to other Artificial Intelligence (AI) alternatives, such as Neural Networks (NNs). Interpretability is especially relevant in control applications, where knowledge of the control equation can be used for evaluating systems' reliability and

* Corresponding author.
E-mail address: francesco.marchetti@dlr.de (F. Marchetti).

https://doi.org/10.1016/j.asoc.2024.111654
Received 15 May 2023; Received in revised form 8 April 2024; Accepted 13 April 2024
Available online 20 April 2024

behavior. For example, in linear systems, the knowledge of the control law expression is used to build the closed-loop transfer function of the whole system [9]. This is then used to perform stability analysis. Moreover, in the context of AI applied to control systems, having an interpretable control law helps increase the trust towards AI-based control systems, making the connection between input and output explicit [10]. The use of GP for control system generation has been previously documented in the literature [11,12]. Yet, it remains relatively infrequent, and to the best of the authors' knowledge, this is the first application of a combination of GP and gradient-descent-based LS to the task of control system design. Specifically, the method developed by Pietropolli et al. [5], named Operators Gradient Descent (OPGD), has been used in this work, considering the promising results reported in its previous application [5]. The idea underpinning OPGD is simple and yet effective: learnable parameters are embedded in GP programs, and the standard GP evolutionary approach is combined with a gradient-based refinement of the individuals. In this study, the method has been adequately modified to deal with control problems. In the original OPGD, the backpropagation technique was employed to evaluate the gradient of the fitness function w.r.t. the GP parameters. The backpropagation is impractical to use in control problems since an implicit dependency between the state and control variables appears in the chain of derivatives. The implicit dependency is caused by the absence of the analytic expression of the states, which results in the impossibility of evaluating, symbolically, the partial derivatives of the states w.r.t. the control variables. Automatic differentiation could be used to avoid the symbolic evaluation of the aforementioned partial derivative, but it would require performing the backpropagation through the ODE solver, which leads to a high memory cost and introduces additional numerical errors [13]. A different approach that would avoid the backpropagation through the solver and that scales efficiently to large problems is the adjoint state method [14] applied in this work. To test the suitability of this OPGD variant, two control problems were chosen: a harmonic oscillator controlled by a Proportional–Derivative (PD) control law and an inverted pendulum on a cart controlled by a Linear Quadratic Regulator (LQR) control law. Experimental results confirm the validity of the proposed algorithm: the produced control laws are well-performing in terms of fitness and control task, and the integration of a local search strategy leads to a substantial improvement in both the desired control structure and the associated parameters compared with others GP-based approaches without any LS mechanism and a feedforward NN.

This paper is structured as follows: Section 2 reviews previous work on combining LS strategies in GP and GP applied to design a control law. Section 3 describes the overall framework introduced in this work, comprising a detailed description of the OPGD technique, IGP algorithm, and adjoint state method, and how they are combined. Subsequently, Section 4 describes the two control problems chosen to test the ability of the GP-based algorithm, and Section 5 discusses the results of the experimental campaign. Finally, Section 6 summarizes the main contribution achieved in this study and provides directions for future works.

2. Related works

This section reviews existing work related to the method developed in this study. In particular, Section 2.1 outlines contributions concerning the combination of GP and local search strategies and then presents recent papers in which gradient-descend-based algorithms have been coupled with the GP evolutionary process. Subsequently, Section 2.2 briefly discusses different approaches for the design of control laws, highlighting the reason for using a GP-based approach in this paper.

2.1. GP with local search and gradient-based algorithms

A refinement process consists of embedding a LS strategy in the evolutionary process. In particular, the additional LS operator considers one or more individuals and searches for the local optima near them. These techniques are a simple type of memetic algorithm [15], which exploits the fact that Evolutionary Algorithms (EAs) can explore large areas of the search space while local optimizers improve solutions gradually and steadily. Their complementary strengths have inspired a lot of novel research in recent years [3,4,16–18].

While several works linking EAs and LS can be found in the literature, the ones that focus on the combination of GP and LS constitute a limited subset [16]. In Eskridge and Hougen [19], authors introduced the LS directly on the GP crossover operator, named memetic crossover, that allows individuals to imitate the observed success of others. Later, in Wang et al. [20], authors proposed a new GP algorithm with local search strategies, named Memetic Genetic Programming (MGP), for dealing with classification problems. Another example can be found in Muñoz et al. [3], where authors proposed a sequential GP memetic structure with Lamarckian inheritance. In this case, two LS methods have been combined: a greedy pruning algorithm and least squares parameter estimation.

Focusing on the combination of GP and gradient-descent-based algorithms, examples can be found in the literature [5,6,21,22]. Nevertheless, existing contributions deal with a specific task or focus on particular components of the evolutionary search. For instance, in Topchy et al. [21], the authors complemented a genetic search for tree-like programs at the population level with terminal values optimization via gradient descent at the individual level. Experimental results show that tuning random constants, besides improving fitness results, requires minimal computational overhead. Zhang and Smart [6] applied a gradient descent algorithm to the numeric parameter terminals in each individual program for object classification problems. Two methods (an online gradient descent scheme and an offline gradient descent scheme) are developed and compared with the basic GP. Experimental results demonstrated that introducing this kind of LS outperforms standard GP in terms of classification accuracy and training time. Another application dealing with constant values optimization can be found in Graff et al. [23], where authors considered the problem of time series forecasting, specifically wind speed time series.

The first example of the inclusion of weight parameters at the internal nodes level is described in the work of Smart and Zhang [24]. Here, a parameter called the inclusion factor is assigned to each node, and a gradient descent search is applied to the inclusion factors. This method obtained promising results, but the experimental study only considered classification tasks. Moreover, the GP system was evaluated using an unusually narrow function set (only sum and multiplication), which is an unrealistic configuration. Later, in Kommenda et al. [25], a gradient-based non-linear least squares optimization algorithm, i.e., Levenberg Marquardt, is used for adjusting constant values in symbolic expression trees during their evolution. Additionally, artificial nodes are inserted in the symbolic expression tree to account for the linear scaling terms.

In Trujillo et al. [17], a Lamarckian memetic GP incorporates LS strategy to refine GP individuals. A simple parametrization for GP trees, where the same functions share the same coefficients, is proposed with different heuristic methods to determine which individuals should be subject to the LS. More recently, in Harrison et al. [26], authors investigate how gradient-based techniques can optimize coefficients in symbolic regression tasks. Lastly, in Pietropolli et al. [5], the authors proposed embedding learnable parameters in GP programs and combining the standard GP evolutionary approach with a gradient-based refinement of the individuals employing the Adam optimizer. Two different algorithms (that differ in how these parameters are shared in the expression operators) are proposed and subsequently tested on real-world problems, demonstrating proficiency in significantly outperforming plain GP.

Due to its simplicity, this GP tree embedding can be easily integrated into other GP approaches, as done in this work. Specifically, in this study, this LS strategy has been applied to a variant of GP, namely the IGP developed by Marchetti and Minisci [27]. IGP was specifically developed for control problems, where it is used to design a control law. Its peculiarity is the capability to promote and maintain population diversity during the evolutionary process. Moreover, it proved superior to a standard GP algorithm, both on control law design and regression tasks. A more detailed description of the IGP is provided in Section 3.2.

2.2. GP and other AI-based approaches for the control laws design

The use of GP to design a control law is not novel in the literature. Koza himself [28] pointed out the capability of GP to automatically design human competitive control laws. Other recent examples can be found in the work of Verdier and Mazo, Jr. [29], where GP is employed to automatically produce a control Lyapunov function and the modes of a switched state feedback controller. In Łapa et al. [30], the authors applied GP to evolve a Proportional Integral Derivative (PID) based controller resistant to noise. To this end, they used a Genetic Algorithm (GA) to optimize the parameters in a GP control law. Another interesting example of GP based Symbolic Regression (SR) used to design a controller is presented in the work of Danai and La Cava [31], where the authors applied a variant of GP, the Epigenetic Linear Genetic Programming (ELGP), to produce the models describing the open-loop input for a desired plant output. This inverse solution approach allows for avoiding the time-consuming closed-loop controller evaluation by performing algebraic evaluations. Diverging from the approaches highlighted in the aforementioned works, the method described in this manuscript employs a gradient-based LS technique relying on the adjoint state method for gradient computation. This methodology generates optimal control laws both in terms of shape and parameters. Moreover, this combination results in reduced computational times compared to the utilization of a GA for the LS phase. Additionally, as described in the subsequent sections, this approach is more suitable than backpropagation for implementing LS within a control environment and represents a novelty in the literature.

Aside from GP, other AI techniques can be used to design control laws, for example, NNs. Plenty of research exists on this topic. Some early applications are described in the book of Irwin et al. [32], while a recent survey of NN control systems applied to aerospace vehicles can be found in [33]. A work closely related to this work is AI Pontryagin of Böttcher et al. [34]. AI Pontryagin is an NN-based control framework capable of designing optimal control laws. Nonetheless, the models produced by this method are not interpretable. Because of the lack of interpretability produced by NNs or other AI algorithms, a thorough comparison of the proposed approach with these techniques was not performed. In fact, the objective of this study is to produce interpretable control laws that resemble the optimal control law of reference, both in terms of shape and parameters.

Nonetheless, alternative AI-based approaches for designing interpretable control laws are documented in the literature. Notably, Hein et al. have undertaken a series of studies incorporating Reinforcement Learning (RL) combined with GP [35–37]. In [35], they introduced the Fuzzy GP Reinforcement Learning (FGPRL) algorithm, utilizing GP to generate Fuzzy Logic (FL) control policies within an RL framework, while [36] explores the use of GP to directly learn algebraic control policies in an RL framework. Lastly, [37] presents a comparative analysis of these approaches against traditional PID and LQR control schemes, as well as other non-conventional methodologies.

Several differences emerge between the proposed work and the approaches presented by Hein et al. Primarily, the learning framework is different, as they evaluated the fitness of the individuals within an RL context. To this end, they generated a database of transition tuples and then used a NN to create a surrogate model of the environment. They showed that GP can effectively learn state–action correlations within this framework. Conversely, the proposed methodology employs a quadratic objective function to evaluate the entire trajectory derived from a GP-based control policy, simulating a trajectory using an available analytical model to directly assess the GP model's performance. While Hein et al.'s approach is well-suited to systems lacking analytical models, their research acknowledges the limitations of directly applying GP to data, as evidenced in [36], where such application results in diminished performance. Contrarily, the findings of this work indicate that integrating the impact of the control policy on the generated trajectory into fitness assessment enables GP to effectively learn a control policy. Furthermore, the goal of this work is to develop control policies that are optimal in both structure and parameters through the application of gradient-based local search to refine the GP models during and after the evolutionary phase. This aspect is only partially addressed in the work of Hein et al. where the emphasis primarily lies on creating structurally optimal models. However, in [35], a local search is performed at the end of the evolutionary process to fine-tune the parameters of the generated FL control policy. It can be argued that performing LS only at the end of the evolutionary process may yield suboptimal results. This is motivated by the observation that poorly performing individuals may result from suboptimal parameter settings. Hence, in this work, it is proposed that these parameters be adjusted throughout the evolutionary process to facilitate more effective exploitation.

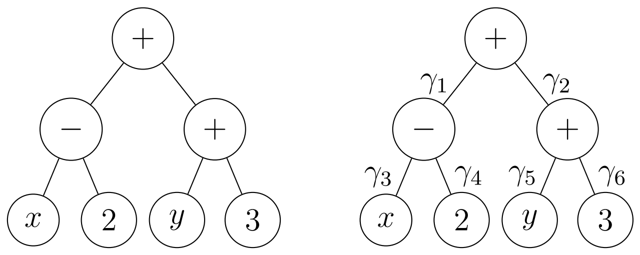


Fig. 1. Depiction of the plain and parametrized GP tree.

3. Parametrized GP with adjoint state method

This Section contains a detailed explanation of the building blocks forming the OPGD-IGP algorithm introduced in this work. The OPGD and IGP algorithms are described along with a detailed discussion on gradient evaluation techniques, justifying the choice of the adjoint state method. This Section concludes with a schematic summary of the overall framework.

3.1. Parametrized genetic programming

One of the main strengths of GP is the possibility of interpreting the solutions that it generates. Nevertheless, the search performed by a GP algorithm only relies on syntactic operations, such as crossover and mutation, to improve the quality of the individuals. In fact, standard GP does not adjust the (implicit) parameters of the given expression. To overcome this problem, different possibilities for integrating a LS algorithm in the GP routine have been proposed in recent years. In this work, the expressive capability of GP individuals is enhanced by adding learnable parameters on their operators, as proposed in [5]. The resulting GP individuals are interpretable as parametric functions, which can be optimized. A canonical GP individual can be represented as a tree where all the edge connections between nodes take a constant value of 1. Yet, the possibility of modifying those values leads to a large spectrum of possible solutions. An example follows.

Fig. 1 shows, on the left, a canonical GP individual encoding the expression in Eq. (1):

$$(x-2)+(y+3) \tag{1}$$

On the right, the same GP individual is enriched with the addition of the parameters γ_i over all the edge connections and encodes the expression in Eq. (2):

$$\gamma_1 \cdot (\gamma_3 \cdot x - \gamma_4 \cdot 2) + \gamma_2 \cdot (\gamma_5 \cdot y + \gamma_6 \cdot 3) \tag{2}$$

Eq. (2) would correspond to Eq. (1) if all the weights γ_i were set to 1.

The Operators Gradient Descent (OPGD) [5] is used, which assigns a different set of weights to each instance of the GP operators, leading to a total number of parameters equal to the number of nodes in the tree. Moreover, to fully exploit the LS potential, a gradient-based optimization of the parameters is performed both during and after the evolutionary process. When the optimizer is used after each generation, it is applied to the whole population of individuals. On the other hand, when applied at the end of the evolutionary process, it is used solely on the best individual of the population obtained. This optimization can be performed using different optimization algorithms, both local and global. The algorithms employed in this work are Adam [38] (during the evolutionary process) and the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm [39] at the end of it. Adam was chosen to achieve faster optimizations during the evolutionary process, while BFGS is preferred at the end of the evolution to better improve the partial results obtained during the evolution. The overall evolutionary process enhanced with the OPGD approach is summarized in Algorithm 1.

Algorithm 1 Pseudocode of evolutionary process with OPGD approach

```
1: Initialize population
2: Store best individual
3: for $i = 1 \rightarrow N_{generations}$ do
4:     for $j = 1 \rightarrow N_{individuals}$ do
5:         Insert learnable parameters in j-th individual
6:         Perform optimization of j-th individual, using Adam with a learning rate $\alpha$ for $n_{opt}$ steps.
7:         Assign the highest fitness found at the previous step to the j-th individual
8:         Remove learnable parameters from j-th individual
9:     end for
10:    Perform crossover and mutation to generate offspring
11:    Evaluate fitness of offspring repeating lines 4 to 9.
12:    Apply selection to generate new population
13:    Update best individual
14: end for
15: Insert learnable parameters in the best individual from all generations
16: Optimize the best individual with BFGS
```

In the original OPGD approach, Adam was used in combination with the backpropagation technique to evaluate the gradient, and this resulted in a fast optimization process. Nonetheless, as explained in Section 3.3, a more suitable approach to evaluate the gradient can be used when dealing with control problems.

3.2. *Inclusive genetic programming*

OPGD can be applied to any GP formulation. In this work, it is applied to the Inclusive Genetic Programming (IGP) introduced in [27]. The resulting method is referred to as OPGD-IGP. IGP was chosen because it was developed specifically for control applications and showed superior performance than standard GP thanks to its ability to promote and maintain the genotypic population's diversity.

Greater genotypic diversity means that bigger individuals are not discarded by the bloat control operators but considered during the crossover and mutation operations, and only the selection is performed to favor smaller individuals. The genotypic material of these bigger individuals may capture the nonlinearities of the studied dynamical system better than smaller individuals, and it is thus an essential piece of information. IGP applies a modified version of the $\mu + \lambda$ evolutionary strategy, the Inclusive $\mu + \lambda$ summarized in Algorithm 2. The core operations are the niches' creation mechanism, the Inclusive Crossover and Mutation, and the Inclusive Tournament. A detailed description of each of these operations is given in [27]. Briefly, a newly created population is subdivided into niches, which act as containers for individuals with a determined size. The maximum and minimum size that a niche can contain is defined by linearly dividing the interval between the maximum and minimum size of the individuals in the population by $n + 1$, where n is the number of niches. The Inclusive Crossover and Mutation consist of applying crossover and mutation by selecting individuals from different niches, and the Inclusive Tournament is a Double Tournament sequentially applied to each niche. Using these operations, a wider distribution of individuals' lengths is considered, and genotypic information is not lost during the evolutionary process due to bloat control operators.

Algorithm 2 Pseudocode of Inclusive $\mu + \lambda$ evolutionary strategy

```
1: Perform population initialization
2: Best individual all-time ← Best individual initial population
3: for $i = 1 \rightarrow N_{generations}$ do
4:     Generate $n$ niches from the current population
5:     Perform Inclusive Crossover and Inclusive Mutation to generate $\lambda$ offspring from $\mu$ parents
6:     Apply Inclusive Tournament to select $\mu$ individuals from a starting population of $\mu$ parents + $\lambda$ offspring
7:     if Fitness of Best individual in $population_i$ > Fitness of Best individual all-time then
8:         Best individual all-time ← Best individual $population_i$
9:     end if
10: end for
```

3.3. *Gradient evaluation techniques*

Gradient-based search algorithms perform the gradient evaluation during the optimization process. The gradient can be evaluated with different approaches, the most common of which is the finite differences approach, which gives a numerical approximation of the gradient at a computational cost proportional to the problem's dimensionality (i.e., the number of optimization variables). For limiting the computational cost, other approaches are employed, such as the backpropagation algorithm [40] often used to optimize a NN's parameters. Backpropagation efficiently computes chain of partial derivatives of the entire NN model [41], leading to a straightforward evaluation of the gradient. However, as explained in the following, backpropagation becomes impractical in control problems. The adjoint state method [14] is chosen as an alternative to evaluate the gradient in the OPGD algorithm applied to a control problem. An additional benefit of this technique is that it can scale efficiently to problems with a high number of optimization variables.

The rest of this Subsection contains a brief demonstration of why the backpropagation approach is impractical in control problems and a description of the adjoint state method.

3.3.1. *Backpropagation*

The backpropagation algorithm is an efficient approach to evaluating derivatives by leveraging the chain rule. In classical regression problems, it is possible to build the entire chain of derivatives to express the gradient of an objective function J with respect to the optimization variables $\boldsymbol{\gamma}$. As an example, a regression problem is considered, and GP is used to create a regression model. The considered GP individual is a function of the selected features and a set of parameters $\boldsymbol{\gamma}$, as described in Section 3.1. The goal is to find the optimal set of parameters $\boldsymbol{\gamma}$ such that an objective function J is minimized. J can be evaluated as the Mean Square Error (MSE) between the output of the GP model $\hat{z}$ and the desired output z, as $J = \frac{1}{n}\sum_{i=1}^{n}(z_i - \hat{z}_i)^2$, where n is the number of samples in the dataset. Using the chain rule, the gradient of J w.r.t. $\boldsymbol{\gamma}$ can be computed as shown in Eq. (3).

$$\frac{\partial J}{\partial \boldsymbol{\gamma}} = \frac{\partial J}{\partial \hat{z}}\frac{\partial \hat{z}}{\partial \boldsymbol{\gamma}} \tag{3}$$

Since $\hat{z}$ (produced by the GP algorithm) is expressed in symbolic form, the partial derivatives in Eq. (3) can be evaluated analytically. Nonetheless, when considering a control problem, i.e. a dynamical system, an implicit dependency between the state variables and the control variables appears. As an example, a control problem is considered where $\mathbf{u}$ is the vector of control variables and $\mathbf{y}$ is the vector of state variables. The dynamical system is defined by Eq. (4), where GP is used to design the control law.

$$\dot{\hat{\mathbf{y}}} = \mathbf{f}(\hat{\mathbf{y}}(t), \mathbf{u}(t)) = \mathbf{f}(\hat{\mathbf{y}}(t), \mathbf{u}_{GP}(\hat{\mathbf{y}}(t), \boldsymbol{\gamma})) \tag{4}$$

The GP model is expressed as a function of the states $\hat{\mathbf{y}}$ and parameters $\boldsymbol{\gamma}$. The goal is to track a desired trajectory $\mathbf{y}$ by finding the optimal set of parameters $\boldsymbol{\gamma}$ that minimizes the error between the obtained trajectory $\hat{\mathbf{y}}$ and the desired trajectory $\mathbf{y}$. By using the chain rule, Eq. (5) is obtained.

$$\frac{\partial J}{\partial \boldsymbol{\gamma}} = \frac{\partial J}{\partial \hat{\mathbf{y}}}\frac{\partial \hat{\mathbf{y}}}{\partial \boldsymbol{\gamma}} = \frac{\partial J}{\partial \hat{\mathbf{y}}}\frac{\partial \hat{\mathbf{y}}}{\partial u}\frac{\partial u}{\partial \boldsymbol{\gamma}} \tag{5}$$

In Eq. (5), the term $\frac{\partial \hat{\mathbf{y}}}{\partial u}$ represents an implicit dependency since the analytical expression of the states is not known. Derivatives of implicit functions can be computed with two techniques: the implicit function theorem, as detailed in the work of Bell et al. [42], or through numerical methods. Concerning the former approach, Margossian et al. [43] analyzed the application of both the implicit function theorem and the adjoint state method, used in this work, for computing derivatives of implicit functions. They demonstrated that while both methods are applicable to any implicit function, the adjoint method typically offers superior efficiency in implicit function differentiation. In particular, the implicit function theorem enables the calculation of directional derivatives for implicit functions using Fréchet derivatives, whereas the adjoint method directly computes these derivatives without intermediary steps, leveraging the inherent structure of the system. Please refer to [43] for the complete demonstration and discussion.

Regarding the use of numerical methods, the straightforward approach would be to use the finite differences technique, which results in a high computational cost. In fact, if the dynamical system is composed of d differential equations and p optimizable parameters, the cost of applying the finite differences is $\mathcal{O}(d(p+1))$, i.e., $p+1$ Ordinary Differential Equations (ODE) propagations at the cost of d differential equations. Another approach is the continuous local sensitivity analysis, which scales proportionally with the number of optimization parameters and leads to a cost of $\mathcal{O}(dp)$ [44]. A last alternative is the adjoint state method. This algorithm computes one forward pass of the ODE system composed of d differential equations and one backward pass of the adjoint dynamical system composed of p differential equations, one for each optimization variable, leading to a computational cost of $\mathcal{O}(d+p)$. Moreover, the derivatives involved in the adjoint method can be computed symbolically, leading to lower computational errors than other numerical methods [45].

3.3.2. Adjoint state method

The adjoint state method has its roots in optimal control theory. It allows the evaluation of the gradient by defining the Lagrangian of the cost functional and the related adjoint variable. The steps to perform the gradient evaluation with the adjoint state method are described in the following. The derivation of the adjoint state method equations presented in the remaining part of this section is taken from [46] and adapted for the proposed work.

Consider the dynamical system in the form of Eq. (6), where $\mathbf{y}$ are the state variables, $\mathbf{u}$ the control variables, $\boldsymbol{\gamma}$ the optimization variables, and $\mathbf{f}$ is a nonlinear mapping describing the initial value problem with $\mathbf{y}(t=0) = \mathbf{y}_0$ as initial conditions.

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}(t), \mathbf{u}(\mathbf{y}(t), \boldsymbol{\gamma})) \tag{6}$$

The goal of the optimization process is to minimize a functional in the form of Eq. (7). To do so, the gradient of J with respect to $\boldsymbol{\gamma}$ is sought, as illustrated in Eq. (8)

$$J(\mathbf{y}, \boldsymbol{\gamma}) = \int_0^T g\,dt + h(T) \tag{7}$$

$$\frac{dJ}{d\boldsymbol{\gamma}} = \int_0^T \frac{dg}{d\boldsymbol{\gamma}}dt + \frac{dh}{d\boldsymbol{\gamma}}(T) = \int_0^T \left(\frac{\partial g}{\partial \boldsymbol{\gamma}} + \frac{\partial g}{\partial \mathbf{y}}\frac{\partial \mathbf{y}}{\partial \boldsymbol{\gamma}}\right)dt + \frac{dh}{d\boldsymbol{\gamma}}(T) \tag{8}$$

The term $\frac{\partial \mathbf{y}}{\partial \boldsymbol{\gamma}}$ in Eq. (8) cannot be computed analytically due to the implicit relation between $\mathbf{y}$ and $\boldsymbol{\gamma}$. To overcome this issue, the optimization can be framed as an equality-constrained minimization problem by introducing the Lagrangian of the function, as in Eq. (9), with the associate adjoint variable $\boldsymbol{\nu}$.

$$\mathcal{L}(\mathbf{y}, \boldsymbol{\gamma}, \boldsymbol{\nu}) = J(\mathbf{y}, \boldsymbol{\gamma}) + \int_0^T \boldsymbol{\nu}(t)^T\left(\mathbf{f} - \frac{d\mathbf{y}}{dt}\right)dt \tag{9}$$

The gradient of the Lagrangian is then computed as in Eq. (10). The last term in the integral in Eq. (10) can be integrated by part resulting in Eq. (11)

$$\frac{d\mathcal{L}}{d\boldsymbol{\gamma}} = \int_0^T \left(\frac{\partial g}{\partial \boldsymbol{\gamma}} + \boldsymbol{\nu}(t)^T\frac{\partial \mathbf{f}}{\partial \boldsymbol{\gamma}} + \left(\frac{\partial g}{\partial \mathbf{y}} + \boldsymbol{\nu}(t)^T\frac{\partial \mathbf{f}}{\partial \mathbf{y}}\right)\frac{d\mathbf{y}}{d\boldsymbol{\gamma}} - \boldsymbol{\nu}(t)^T\frac{d}{dt}\frac{d\mathbf{y}}{d\boldsymbol{\gamma}}\right)dt + \frac{dh}{d\boldsymbol{\gamma}}(T) \tag{10}$$

$$\begin{aligned}\frac{d\mathcal{L}}{d\boldsymbol{\gamma}} = &\int_0^T \left(\frac{\partial g}{\partial \boldsymbol{\gamma}} + \boldsymbol{\nu}(t)^T\frac{\partial \mathbf{f}}{\partial \boldsymbol{\gamma}} + \left(\frac{\partial g}{\partial \mathbf{y}} + \boldsymbol{\nu}(t)^T\frac{\partial \mathbf{f}}{\partial \mathbf{y}} + \left(\frac{d\boldsymbol{\nu}}{dt}\right)^T\right)\frac{d\mathbf{y}}{d\boldsymbol{\gamma}}\right)dt + \\ &+ \boldsymbol{\nu}(0)^T\frac{d\mathbf{y}}{d\boldsymbol{\gamma}}(0) - \boldsymbol{\nu}(T)^T\frac{d\mathbf{y}}{d\boldsymbol{\gamma}}(T) + \frac{dh}{d\boldsymbol{\gamma}}(T)\end{aligned} \tag{11}$$

Since the optimization problem was rewritten as an equality-constrained optimization, the goal is to set the second term in Eq. (9) to zero, therefore resulting in $\mathcal{L}(\mathbf{y}, \boldsymbol{\gamma}, \boldsymbol{\nu}) = J(\mathbf{y}, \boldsymbol{\gamma})$. According to this, it can be stated that the gradient of the Lagrangian in Eq. (11) corresponds to the gradient of the functional ∇J_{γ}.

By setting some of the elements in Eq. (11) to zero, it can be used to evaluate the gradient of the functional. The resulting set of equations is summarized in Eq. (12).

$$\frac{dJ}{d\boldsymbol{\gamma}} = \int_0^T \left(\frac{\partial g}{\partial \boldsymbol{\gamma}} + \boldsymbol{\nu}(t)^T\frac{\partial \mathbf{f}}{\partial \boldsymbol{\gamma}}\right)dt \tag{12a}$$

$$\frac{d\boldsymbol{\nu}}{dt} = -\left(\frac{\partial \mathbf{f}}{\partial \mathbf{y}}\right)^T\boldsymbol{\nu}(t) - \left(\frac{\partial g}{\partial \mathbf{y}}\right)^T \tag{12b}$$

$$\boldsymbol{\nu}(t=T) = \frac{dh}{d\mathbf{y}}(T) \tag{12c}$$

Employing the notation introduced in [46], and summarized in Eq. (13), Eq. (12) can be simplified as Eq. (14)

$$\mathbf{A} = \frac{\partial \mathbf{f}}{\partial \mathbf{y}}, \mathbf{B} = \frac{\partial \mathbf{f}}{\partial \boldsymbol{\gamma}}, \boldsymbol{\eta} = \frac{\partial h}{\partial \mathbf{y}}, \boldsymbol{\phi} = \frac{\partial g}{\partial \mathbf{y}}, \boldsymbol{\psi} = \frac{\partial g}{\partial \boldsymbol{\gamma}} \tag{13}$$

$$\frac{dJ}{d\boldsymbol{\gamma}} = \int_0^T \left(\boldsymbol{\psi} + \boldsymbol{\nu}(t)^T\mathbf{B}\right)dt \tag{14a}$$

$$\frac{d\boldsymbol{\nu}}{dt} = -\mathbf{A}^T\boldsymbol{\nu}(t) - \boldsymbol{\phi}^T \tag{14b}$$

$$\boldsymbol{\nu}(t=T) = \boldsymbol{\eta}(T) \tag{14c}$$

The overall process of evaluating the gradient using the adjoint state method can be summarized in three steps:

1. Forward propagation of the dynamic system in Eq. (4).
2. Backward propagation of the adjoint system in Eq. (14b) evaluating the initial conditions with Eq. (14c). This propagation is performed from $t = T$ to $t = 0$.

3. Evaluate the gradient with Eq. (14a) and the objective function with Eq. (7)

The adjoint state method can be applied effortlessly to optimize a GP control law. The parametrized GP laws enter the dynamical system as in Eq. (4), where $\boldsymbol{\gamma}$ are the parameters to be optimized or the optimization variables in the optimization problem. The GP law $\mathbf{u}_{GP}$ can be built using only differentiable functions, resulting in a differentiable equation that can be inserted explicitly in the equation of motion. Subsequently, the partial derivatives in Eq. (13) can be evaluated symbolically.

From this point onward, the acronym OPGD will be used to refer to the OPGD with the gradient evaluation performed using the adjoint state method.

3.4. OPGD-IGP framework summary

Fig. 2 presents the frameworks of plain OPGD and OPGD-IGP. The latter represents the novel approach introduced in this work. The novelties introduced in this work are highlighted in Fig. 2 by the colored boxes and are the following: (1) the GP algorithm to which OPGD is applied; (2) the target application or data source; (3) the approach used to evaluate the gradient; (4) the optimizer used to optimize the best individual found during the evolutionary process. The original OPGD was used to enhance a standard GP algorithm, while in this work, it was applied to IGP as highlighted by the *GP algorithm* box. The second difference lies in the data passed to OPGD. In OPGD-IGP, the data originated through the interaction with a dynamical system, while in the original OGPD, a static dataset was used (*Data source* box). Because of this different data source, a different approach to evaluate the gradient is employed (*Gradient evaluation approach* box), as explained in the previous subsections. Lastly, to optimize the best individual found during the evolutionary process, OPGD-IGP relies on the BFGS optimization algorithm, while in the standard OPGD, Adam was employed (*Optimizer* box). In comparison to the broader academic literature, the OPGD-IGP represents a novel approach to control law generation. Traditional methods of deriving control laws through GP typically do not prioritize the attainment of an optimal controller in terms of both its structure and parameters. Conversely, the LS integrated within the OPGD-IGP facilitates the achievement of such optimality. Additionally, in conventional gradient-based LS methodologies, gradient computation often relies on finite differences or backpropagation. The incorporation of the adjoint state method within this framework represents an innovative step forward from these conventional approaches, exhibiting superior suitability for control applications, as demonstrated in this study.

4. Experimental study — test cases

Two control problems are chosen to test the ability of the OPGD-IGP to design a control law automatically, both in terms of shape and parameters: a harmonic oscillator controlled by a PD control scheme and an inverted pendulum controlled by an LQR control scheme.

4.1. Harmonic oscillator

The formulation of the harmonic oscillator is the one defined in [46] and described by the nonlinear ODE system in Eq. (15). The state variables are the position x and the speed v. Thus, $\mathbf{y} = [x, v]$. u is the control variable.

$$\begin{aligned} \dot{x} &= v \\ \dot{v} &= -\frac{k}{m}x - \frac{c}{m}(ax^2 + b)v + \frac{u}{m} \end{aligned} \tag{15}$$

The constant parameters used in Eq. (15) are reported in Table 1. The initial conditions were set as $x_0 = 4$ m, $v_0 = 0$ m/s, $t_0 = 0$ s, while the desired final conditions are $x_f = 0$ m, $v_f = 0$ m/s, $u_f = 0$ N, $t_f = 10$ s.

Table 1
Harmonic oscillator parameters.

Parameter	Value	Description
m	1 kg	Mass
k	2 kg/s^2	Spring stiffness
a	1 m^{-2}	First damper coefficient
b	−1	Second damper coefficient
c	0.3 kg/s	Third damper coefficient

Table 2
Inverted pendulum parameters.

Parameter	Value	Description
M	0.1 kg	Cart mass
m	0.02 kg	Pendulum mass
L	0.1 m	Pendulum length
g	−9.8 m/s^{-2}	Gravitational acceleration

The control scheme designed for this test case is a PD control scheme that receives as input the tracking errors on the position e_x and speed e_v. The methodology used to obtain the proportional and derivative gains is described in [46]. Eq. (16) illustrates the final control law used as a reference.

$$u = -1.753e_x - 3.010e_v \tag{16}$$

4.2. Inverted pendulum on a cart

The formulation of the inverted pendulum was taken from Brunton and Kutz [47] and described by the nonlinear ODE system in Eq. (17). The states variables are the position x, speed v, angular position θ and angular speed ω, therefore $\mathbf{y} = [x, v, \theta, \omega]$. u is the control variable.

$$\begin{aligned} \dot{x} &= v \\ \dot{v} &= \frac{-m^2L^2\ g\cos(\theta)\sin(\theta) + mL^2(mL\omega^2\sin(\theta)) + mLu^2}{mL^2(M + m(1-\cos(\theta)^2))} \\ \dot{\theta} &= \omega \\ \dot{\omega} &= \frac{(m+M)mgL\sin(\theta) - mL\cos(\theta)(mL\omega^2\sin(\theta)) - mL\cos(\theta)u}{mL^2(M + m(1-\cos(\theta)^2))} \end{aligned} \tag{17}$$

The constant parameters used in Eq. (17) are described in Table 2. The initial conditions were set as $x_0 = -1$ m, $v_0 = 0$ m/s, $\theta_0 = \pi$+0.1 rad $\omega_0 = 0$ rad/s, $t_0 = 0$ s, and the desired final conditions are $x_f = 1$ m, $v_f = 0$ m/s, $\theta_f = \pi$ rad, $\omega_f = 0$ rad/s, $u_f = 0$ N, $t_f = 10$ s.

The same LQR design process described in [47] was used, with $\mathbf{Q}$ set as a 4×4 identity matrix and $R = 1$.

The reference control law for the LQR scheme is displayed in Eq. (18), where the input variables are the errors on the states.

$$u = -\mathbf{K}\mathbf{e} = 1e_x + 1.419e_v - 8.131e_\theta - 1.223e_\omega \tag{18}$$

The parameters used to design the LQR controller were chosen to have the LQR gains close to 1. This is necessary to have a good outcome from the optimization process: because a local optimization scheme was employed, the choice of the initial condition influences the optimization process. Since no prior information is available on the initial value of the GP parameters, these were initialized as 1. Therefore, the optimization process converges if the desired value is close to 1 as well. Different optimization approaches can be used to deal also with larger gain values. Nonetheless, it is not the aim of this work to explore different optimization algorithms.

5. Experimental results

To test the proposed methodology, Standard Genetic Programming (SGP), IGP, OPGD-IGP, and a feedforward NN were compared. The computational costs associated with these algorithms are summarized in Table 3. Referring to the terminology employed in Section 3.3.1,

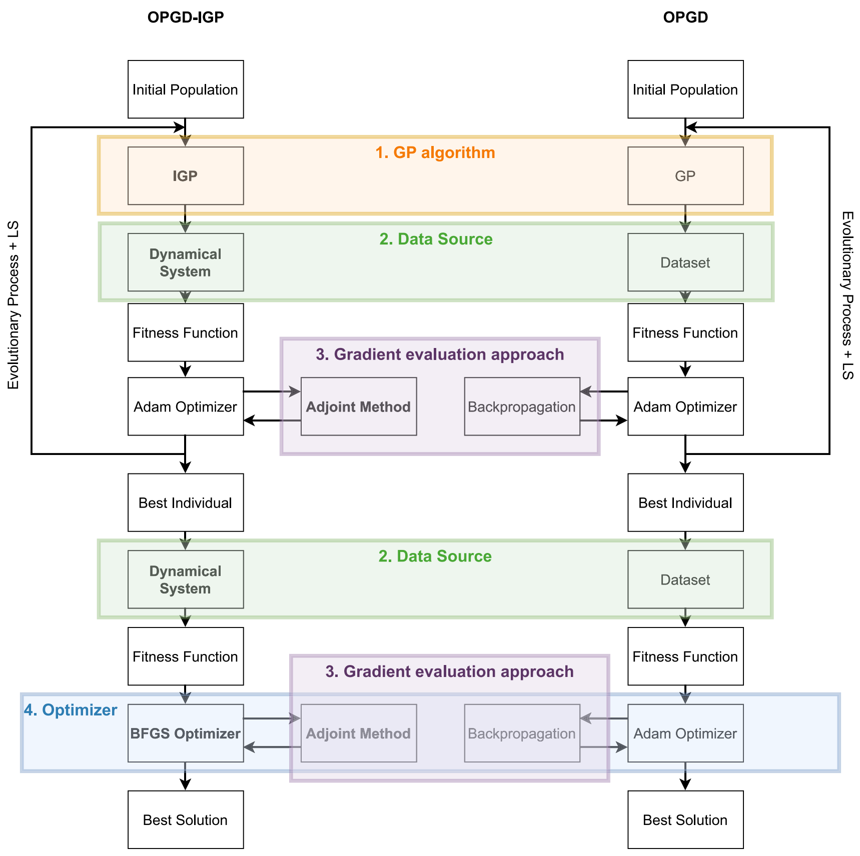


Fig. 2. Diagrams of the OPGD-IGP (left) and OPGD (right) workflows.

Table 3
Computational costs associated with the analyzed algorithms and test cases.

Algorithm	Computational Cost
SGP	$\mathcal{O}(n_g n_i d)$
IGP	$\mathcal{O}(n_g n_i d)$
OPGD-IGP	$\mathcal{O}((n_g n_i n_{opt} + n_{BFGS})(d + p))$
NN Loop	$\mathcal{O}(n_{BFGS}(d + p))$

d is the number of differential equations in the considered dynamical system, while p is the number of optimization variables that correspond to the number of differential equations of the adjoint system. n_g is the number of generations, n_i the number of individuals, n_{opt} the number of intra-evolution optimization steps and n_{BFGS} the number of extra-evolution optimization steps, which correspond to the training epochs for the NN trained in the loop.

The computational cost represents the theoretical cost associated with the complete execution of the algorithm. For the SGP and IGP, this cost is in the order of $\mathcal{O}(n_g n_i d)$, meaning that one trajectory propagation for a dynamical system comprising d differential equations is performed for each individual at each generation. Conversely, for the OPGD-IGP, the computational cost is in the order of $\mathcal{O}((n_g n_i n_{opt} + n_{BFGS})(d + p))$, Here, n_{opt} executions of the adjoint state method, involving one forward propagation of d differential equations and one backward propagation of p differential equations, are performed for each individual at each generation. Then, n_{BFGS} optimization steps are performed at the end of the evolutionary process on the best-performing individual. Regarding the NN trained in the loop, the adjoint state method is applied at each training epoch, resulting in a computational cost in the order of $\mathcal{O}(n_{BFGS}(d + p))$.

The control laws' parameters for the two reference control schemes were obtained through an optimization process. Therefore, the goal of the experimental campaign is to use the aforementioned algorithms to design a well-performing control law by solving the same optimization problem as those considered in the references. The similarity between the obtained control laws and the reference ones, both in terms of shape and parameters, is considered to assess the success of the experiments.

On the other hand, the NN does not produce interpretable models and is only used as a reference to understand how OPGD-IGP compares against a different and more established approach. The NN is trained in two ways: (1) with a dataset produced using the optimal control laws of reference — this experiment is meant to discover the smallest configuration necessary to learn the desired model; (2) training the NN in-the-loop as done with the OPGD-IGP. This last training method is summarized in Algorithm 3.

Algorithm 3 Pseudocode of the training process with NN in-the-loop

```
1: Create NN model
2: Extract the NN weights and store them in the vector of optimization variables $\mathbf{p}$
3: Start optimization process
4: while Termination criteria is not met do
5:     Insert the updated weights from $\mathbf{p}$ into the NN
6:     Propagate the ODE system using the NN as controller
7:     Evaluate the objective function according to the obtained trajectory
8:     Evaluate the gradient with the adjoint state method
9:     Update the $\mathbf{p}$ vector with the optimizer routine
10: end while
```

A discussion of the outcome of each training method is provided at the end of Sections 5.3 and 5.4 for the oscillator and pendulum test cases respectively. A description of the dataset and training results of the former approach is provided in Appendix A. For the SGP, IGP and OPGD-IGP algorithms, 30 independent runs were performed to obtain a statistical sample. The Adam optimizer in OPGD-IGP considered a

Table 4
SGP, IGP and OPGD-IGP settings for both test cases.

	Oscillator	Pendulum
Population Size	300 individuals	
Maximum Generations	300	
Stopping criteria	Reaching maximum number of generations	
Crossover probability	0.2 → 0.65	
Mutation probability	0.7 → 0.25	
Evolutionary strategy	Inclusive $\mu + \lambda$	
μ	Population size	
λ	Population Size × 1.2	
Limit Height	10	
Limit Size	15	30
Selection Mechanism	Inclusive Tournament	
Double Tournament fitness size	2	
Double Tournament parsimony size	1.2	
Tree creation mechanism	Ramped half and half	
Mutation mechanisms	Uniform (50%), Shrink (5%), Insertion (25%), Mutate Ephemeral (20%)	
Crossover mechanism	One point crossover	
Primitives Set	$+, -, \times$	

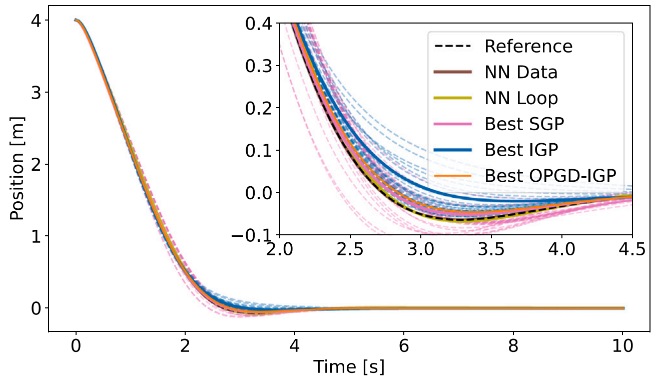


Fig. 3. Harmonic oscillator's position x trajectories obtained using SGP, IGP, OPGD-IGP, and the NN models.

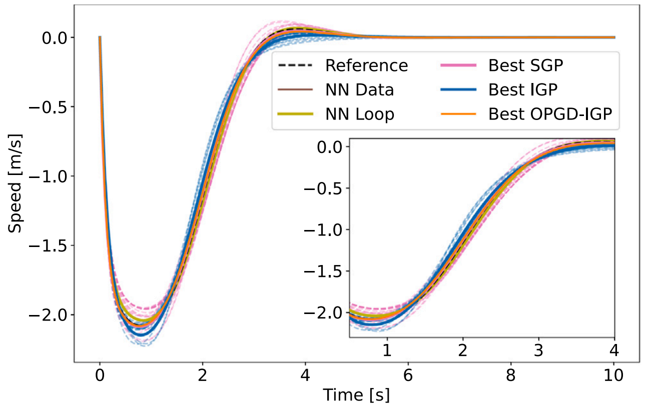


Fig. 4. Harmonic oscillator's speed v trajectories obtained using SGP, IGP, OPGD-IGP, and the NN models.

learning rate of 0.01 and 5 optimization steps, respectively α and n_{opt} in Algorithm 1, during the evolutionary process. At the end of the evolutionary process, the best individual is optimized using the BFGS algorithm implemented in the Python library Scipy [48]. An objective function precision threshold of 10^{-6} was used as termination criterion for the BFGS algorithm. BFGS with these settings was used to train the NN in-the-loop as well. The developed code will be available at https://github.com/strath-ace/smart-ml.

5.1. GP settings

The common settings of OPGD-IGP, IGP, and SGP for the two test cases are listed in Table 4. IGP and SGP use two ephemeral constants, while OPGD-IGP does not consider ephemeral constants. This choice is motivated by the fact that OPGD-IGP should be able to find the correct parameters autonomously. On the other hand, ephemeral constants are necessary to allow IGP and SGP to evolve parametric control laws. Differently from IGP, SGP uses the standard $\mu + \lambda$ evolutionary strategy and the Double Tournament selection process. Finally, the SGP crossover and mutation probability are fixed respectively to 0.8 and 0.2. All GP algorithms receive as input the tracking errors on the states and output the control force u.

5.2. Fitness function

For the two test cases, the fitness function was computed as $F = -J$, where J is detailed in Eq. (7). This adjustment is made to ensure consistency in terminology, given that fitness is a metric intended for maximization. Conversely, the selected objective function J is designed for a minimization problem, and the comparison with the reference control schemes is based on the objective function value. Therefore, the discussion presented in the following will refer to the objective function rather than the fitness. The functions g and h in Eq. (7) are set as quadratic functions, as described in Eqs. (19) and (20),

$$g = \frac{1}{2}(\mathbf{e}_y^T \mathbf{Q}_g \mathbf{e}_y + \mathbf{e}_u^T \mathbf{Q}_u \mathbf{e}_u) \tag{19}$$

$$h = \frac{1}{2}\mathbf{e}_y^T \mathbf{Q}_h \mathbf{e}_y \tag{20}$$

where $\mathbf{e}_y$ is the vector of the tracking errors on the state variables, and $\mathbf{e}_u$ is the vector of the tracking errors on the control variables. $\mathbf{Q}_g, \mathbf{Q}_u, \mathbf{Q}_h$ are diagonal matrices used to weight the different contributions to the objective function. These functions are used to minimize the tracking errors on the states and control variables. Using Eq. (7), the integral of g is evaluated, leading to the minimization of both the states and controls tracking errors on the whole trajectory. h is used to evaluate the tracking error on the final position. In this work, also the tracking for the complete trajectory is performed against the desired final conditions. Therefore, each reference trajectory can be imagined as a constant line at the desired value of the considered state or control variable.

5.3. Harmonic oscillator

For this test case, the objective function's parameters were set as follows: $\mathbf{e}_y = [e_x, e_v]$, $\mathbf{e}_u = e_u$, $\mathbf{Q}_g = diag([5,5]), \mathbf{Q}_u = 1, \mathbf{Q}_h = diag([1,1])$. The tracking errors are evaluated as the difference between

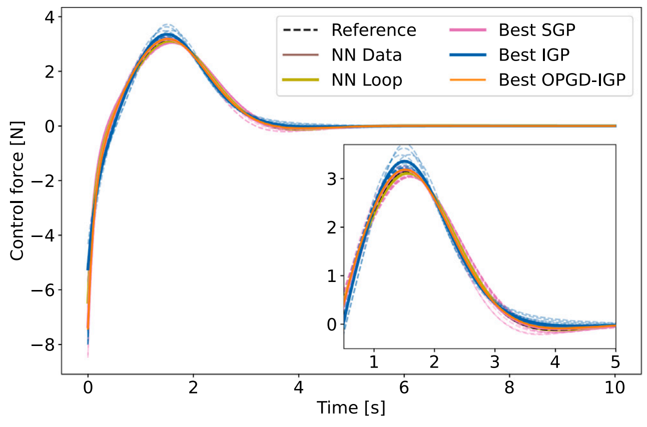


Fig. 5. Harmonic oscillator's control action u trajectories obtained using SGP, IGP, OPGD-IGP, and the NN models.

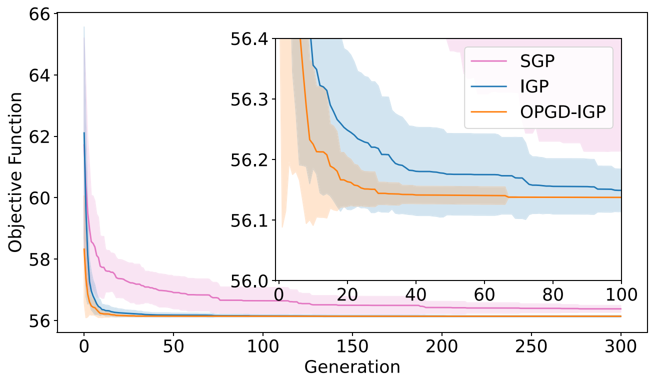


Fig. 6. Objective function evolution of the SGP, IGP, and OPGD-IGP algorithms for the harmonic oscillator case. The solid lines represent the mean, while the shaded areas depict the error bands, i.e. standard deviations, for both algorithms.

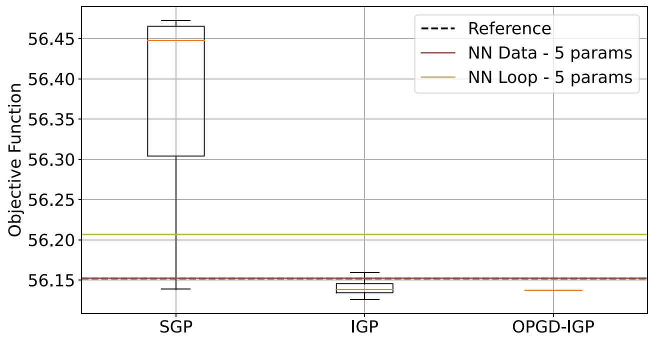


Fig. 7. Objective function of the best-performing individual for the SGP, IGP, OPGD-IGP, and NN models for the harmonic oscillator case. For the GP-based algorithms, 30 simulations were considered.

the current state and control variables and the desired final values listed in Section 4.1. Using the reference control law, a reference objective function equal to $J = 56.152$ was obtained by applying Eq. (7). The objective function was evaluated by propagating the dynamical system using the Runge–Kutta 4 scheme with an integration step of 0.05 s.

The obtained results are presented from Figs. 3 to 7 and in Appendix B. In the following Figures, *NN Data* refers to the NN trained on the dataset, while *NN Loop* refers to the NN optimized in the control loop. The smallest NN architecture capable of capturing the optimal control law behavior is composed of one hidden layer with one neuron. More details are given in Appendix A. The same configuration is trained in-the-loop to compare the effect of a different training approach.

Figs. 3 and 4 depict the state trajectories, while Fig. 5 shows the control force trajectories. In these plots, the reference trajectory, obtained via the reference control law, is depicted as a dashed black line. The pink lines represent the trajectories obtained with SGP, the blue lines represent those obtained with IGP, the orange lines represent those obtained with OPGD-IGP, while the brown and olive lines the trajectories obtained with the NNs trained on the dataset and in-the-loop, respectively. For SGP, IGP, and OPGD-IGP, the continuous lines show the best of the 30 runs performed while the dashed lines represent trajectories from all the other runs. The inset in each plot highlights the distribution of the obtained trajectories. As can be seen in Figs. 3, 4 and 5, all the tested algorithms evolve well-performing control laws, capable of generating a behavior close to the reference one. Among the GP algorithms, it can be seen how SGP is the least consistent, with many of the produced trajectories straying from the reference. When considering IGP and OPGD-IGP, the magnified sections show that IGP produces control laws that result in a wider range of behaviors, while the trajectories produced with OPGD-IGP are all overlapped, meaning that they always converge to the same mathematical model. Regarding the NN models, the NN Data trajectory is not clearly visible since it perfectly overlaps with the reference one, whereas the NN Loop trajectory is close to the reference with a behavior similar to the best of the OPGD-IGP trajectories.

Figs. 6 and 7 depict statistical analyses of the obtained objective function values. Specifically, Fig. 6 shows the best individual's objective function evolution. It is possible to observe that OPGD-IGP can reach the final solution in fewer generations ($\sim$ 65 generations) with respect to IGP ($>$ 100 generations), while SGP results are worse than the other two GP-based algorithms.

Fig. 7 displays the objective function values obtained in the simulations performed with the GP algorithms and both NN's training approaches. The objective function of the NN trained in-the-loop comes naturally from the optimization process, while the objective function of the NN trained on the data is obtained by propagating a trajectory with the trained model and evaluating the objective function as described in Section 5.2.

Looking at Fig. 7, it can be seen how OPGD-IGP always converges to the same individual, while IGP tends to produce different control laws with different performance and, as observed also from Figs. 3 to 5, SGP is the least consistent performer among the GP algorithms. Moreover, these boxplots show that IGP can achieve a lower objective value than OPGD-IGP. This is likely due to the random mutation applied to the ephemeral constants. This mechanism, absent in OPGD-IGP, allows for a greater exploration of the search space in contrast to the exploitation fostered by the use of LS. Regarding the NNs, it can be seen how the two training approaches lead to slightly different results. In fact, the objective function obtained with the NN trained on the data matches almost exactly the reference objective function, while the one trained in-the-loop shows an objective function worse than IGP and OPGD-IGP. This suggests that a network with more parameters is required to improve the results with the train in-the-loop approach.

The complete list of models produced by the GP algorithms can be found in Appendix B. As one can observe, IGP and SGP produce a variety of models, while OPGD-IGP always converges to the same combination of control law shape and parameters, thus confirming the ability of OPGD-IGP to autonomously produce the desired control law for a dynamical system in terms of shape and parameters. Table 5 lists the reference control law and the most frequent OPGD-IGP control law. The difference between the reference and the obtained optimal parameters is caused by the different optimization algorithms used in this work and in [46].

5.4. Inverted pendulum on a cart

For this test case, the objective function's parameters were set as follows: $\mathbf{e}_y = [e_x, e_v, e_\theta, e_\omega]$, $\mathbf{e}_u = e_u$, $\mathbf{Q}_g = diag([5, 5, 5, 5])$, $\mathbf{Q}_u = 1$, $\mathbf{Q}_h =$

Table 5
Reference control law and most frequent model output by the OPGD-IGP for the harmonic oscillator test case.

	Control Law
Reference	$-1.753e_x - 3.010e_v$
OPGD-IGP	$-1.854e_x - 3.158e_v$

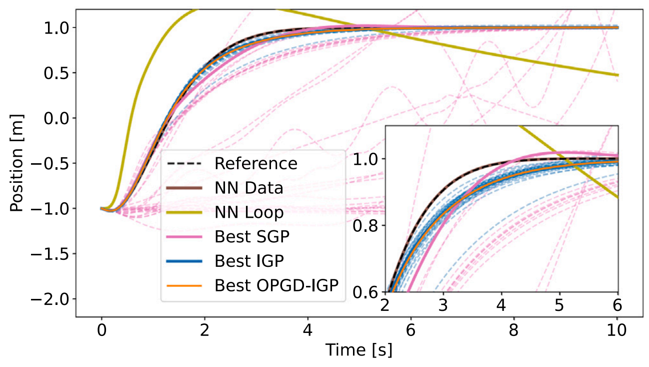


Fig. 8. Trajectories of the pendulum's position x obtained using SGP, IGP, OPGD-IGP and the NN models.

$diag([1, 1, 1, 1])$. The tracking errors are evaluated between the current and the desired final values reported in Section 4.2. The optimization problem is structured in a slightly different way than the reference. The same objective function is used, but different plant models are employed. In particular, the reference control law was evaluated using the linearized models necessary to perform the LQR design while SGP, IGP,OPGD-IGP, and NNs were tested using the complete nonlinear model in Eq. (17). This procedure allows for assessing the ability of the tested algorithms to produce the desired control law when considering a nonlinear model.

As for the previous test case, in Figs. 8 to 14 *NN Data* refers to the NN trained on the dataset, while *NN Loop* refers to the NN optimized in the control loop. Again, the smallest NN architecture capable of capturing the optimal control law behavior consists of one hidden layer with one neuron. More details are given in Appendix A. As for the oscillator case, the same configuration is trained in-the-loop to assess the effect of a different training approach.

Using the reference control law, an objective function value $J = 16.264$ is obtained. The objective function is evaluated by applying Eq. (7) after propagating the dynamical system using the Runge–Kutta 4 scheme with an integration step of 0.01 s. The integration step was reduced to 0.005 s to perform the training of the NN in-the-loop (more details at the end of this subsection.).

Figs. 8 to 12 depict the trajectories obtained by propagating the dynamical system using the control laws evolved by the SGP, IGP, and OPGD-IGP algorithms on the 30 simulations performed and by the best-performing NN architectures (obtained using both training approaches). As for the previous test case, the continuous lines represent the best solution, while the dim dashed lines depict the other ones. The black dashed line represents the reference trajectory. These results prove the capability of IGP, OPGD-IGP, and the NN trained on data to produce well-performing control laws, while SGP and NN trained in-the-loop show poorer performance. Once again, OPGD-IGP performs more consistently than IGP, producing a set of overlapping trajectories. On the other hand, IGP and SGP produce a broad range of models, some of which do not exhibit good performance in terms of trajectory. Regarding the NN results, the training performed on the data led to a perfect overlap with the reference trajectory, while the training in-the-loop failed to find a well-behaving model.

Figs. 13 and 14 show the statistical analysis of the objective function values. As for the oscillator test case, Fig. 13 highlights the faster

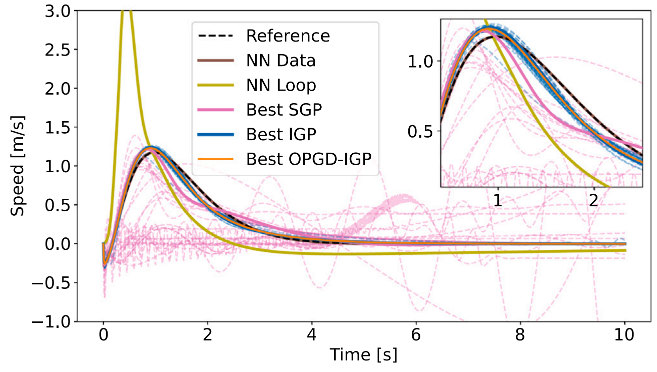


Fig. 9. Trajectories of the pendulum's speed v obtained using SGP, IGP, OPGD-IGP and the NN models.

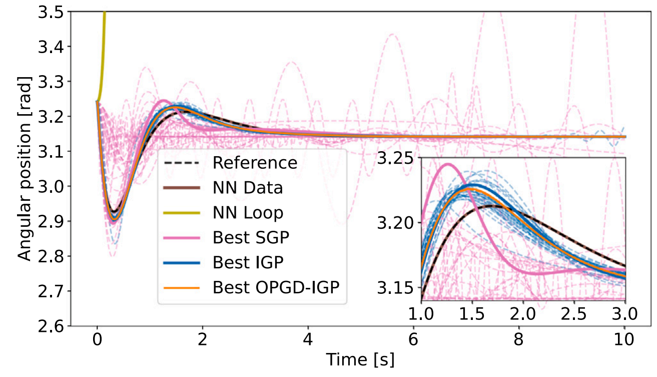


Fig. 10. Trajectories of the pendulum's angular position θ obtained using SGP, IGP, OPGD-IGP and the NN models.

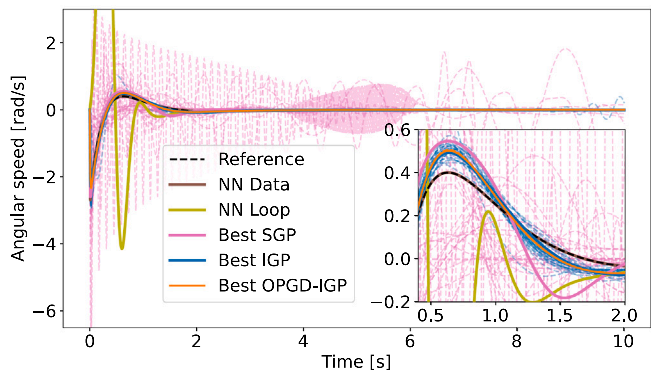


Fig. 11. Trajectories of the pendulum's angular speed ω obtained using SGP, IGP, OPGD-IGP and the NN models.

convergence of OPGD-IGP compared to IGP. OPGD-IGP can reach the minimum objective function in ~ 40 generations while IGP requires more than 100 generations. As for the previous test case, SGP performs worse than the other two GP algorithms.

Fig. 14 displays the statistical distribution of the objective function values obtained with the tested algorithms. The boxplots for the GP algorithms are created considering the best objective function value achieved in each of the 30 simulations. OPGD-IGP converges to similar individuals and also reaches a lower objective function compared to the IGP algorithm, while the SGP produced individuals with worse performance than the other GP algorithms. Regarding the NN results, the training from data led to an almost perfect match with the optimal solution, while the training in-the-loop led to poor results. These results are discussed in Section 5.5.

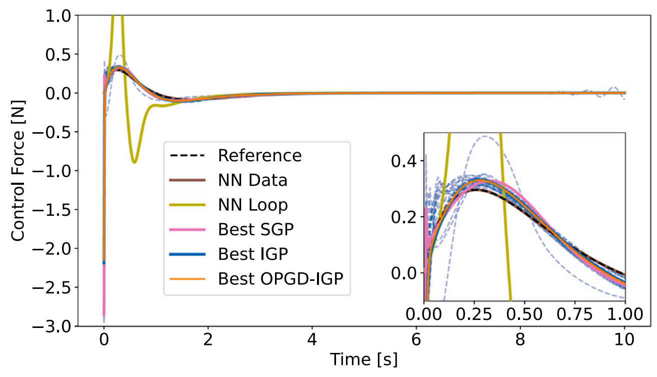


Fig. 12. Trajectories of the pendulum's control force u obtained using SGP, IGP, OPGD-IGP and the NN models.

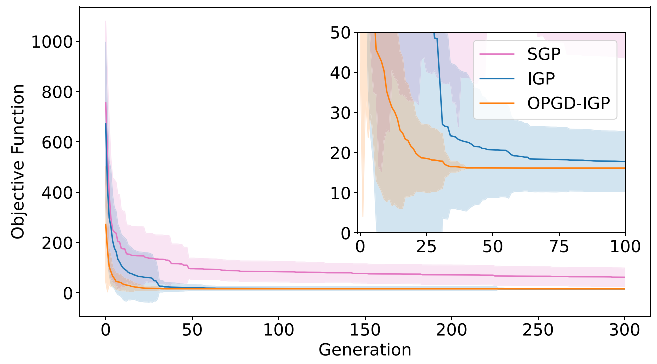


Fig. 13. Objective function evolution of the SGP, IGP and OPGD-IGP algorithms for the pendulum case. The solid lines represent the mean, while the shaded areas show the error bands, i.e. standard deviations.

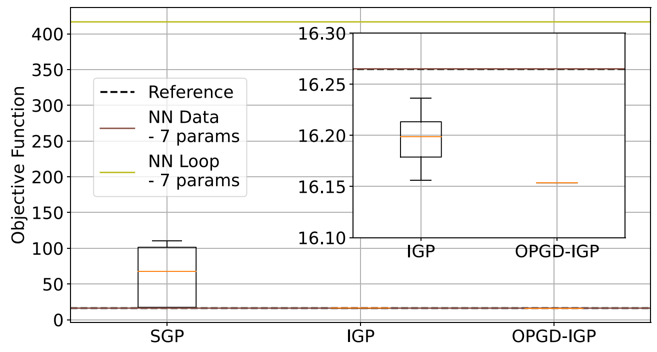


Fig. 14. Objective function of the best-performing individual for the SGP, IGP, OPGD-IGP and NN models for the pendulum case. For the GP algorithms, 30 simulations were considered.

The complete list of the models produced by the GP algorithms is listed in Appendix B. These results show that OPGD-IGP can often (21/30 simulations) converge to individuals with the same shape and similar parameters to the reference one. On the other hand, IGP produces only one model (simulation 5) with the same shape and similar parameters as the reference. SGP is capable of finding more models with the appropriate shape than IGP. However, the parameters are far from their optimal values (e.g., simulations 7, 10, 11, 24, 27), and the overall result is a set of models that perform worse than those found by IGP.

Table 6 lists the reference control law and the most frequent model obtained by the OPGD-IGP method.

The difference in the parameters' values is caused by the difference in the plant's models used to obtain them and the employed

Table 6
Reference control law and most frequent model output by OPGD-IGP for the inverted pendulum test case.

	Control Law
Reference	$1e_x + 1.419e_v - 8.131e_\theta - 1.223e_\omega$
OPGD-IGP	$0.781e_x + 1.161e_v - 5.842e_\theta - 0.952e_\omega$

optimization schemes. The LQR gains are evaluated by solving the continuous-time algebraic Riccati equation using the linearized plant models. On the other hand, in OPGD-IGP, the parameters are optimized with a numerical scheme, and the complete nonlinear models are considered. This result is particularly interesting since it showcases how OPGD-IGP can be applied to a fully nonlinear model and still produces a control law close to the optimal one. This approach would allow designing an optimal control law even for complex systems that cannot be linearized or without resorting to linearization techniques that can cause a loss of information.

5.5. *Summary of findings*

The conducted experiments yielded several observations. Firstly, IGP consistently outperforms SGP, providing further evidence of its suitability for the task of designing control schemes. In turn, IGP is outperformed by OPGD-IGP, which incorporates the LS strategy. The latter shows superior performance and statistical consistency compared to the original IGP, consistently producing control laws that closely match the reference ones in terms of shape, albeit with minor differences in terms of parameters. These differences are due to the different optimization algorithms employed (Fletcher-Reeves vs. Adam and BFGS), as observed in the oscillator case, and differences in the employed plant models (linear vs. nonlinear), as seen in the pendulum case.

The ability to generate optimal control laws is only partially observed in the other two GP algorithms. Regarding the oscillator case, they can produce models with similar shapes but with randomly assigned parameters, resembling the reference model but lacking consistency across multiple runs. In the pendulum case, they fail to achieve the desired shape since the problem's increased complexity forces the GP algorithms to generate more complex models to compensate for suboptimal parameters. Furthermore, the convergence speed benefits from the embedded LS strategy, enabling OPGD-IGP to converge in approximately half the number of generations required by IGP alone.

This comparison highlights the role of LS in producing optimal control laws in terms of both shape and parameters. Furthermore, it illustrates how LS can enhance the convergence properties of GP algorithms.

Regarding the NN models, two different training methodologies were tested for the NN: (1) training with a dataset generated from the reference optimal control law and (2) training within the control loop. The first approach primarily serves to determine the minimal configuration capable of learning the reference optimal control law. In fact, this approach cannot be directly compared with the OPGD-IGP since the latter learns how to control a system by interacting with it, while the NN trained on pre-existing data lacks knowledge of the system to be controlled. Furthermore, if the control law is available and used to produce the dataset, creating a regression model on those data becomes superfluous.

The objective of OPGD-IGP is to generate an interpretable control law, similar to the optimal one both in shapes and parameters, by solving the same optimization problem used to find the reference control law. Consequently, this study aims to demonstrate that the OPGD-IGP can autonomously find an interpretable and optimal control law solely by interacting with the controlled system knowing only the high-level goal, i.e., the objective function of the optimization problem, and with no prior knowledge of the reference control law itself. That is why the

NN is also trained in-the-loop, i.e., in the same training setting used by OPGD-IGP. The smallest configuration found after training on the data is considered since it proves that the NN has enough parameters to learn the desired control law. Thus, it should also be able to do it when trained in-the-loop. While this is true in the oscillator case, where the two training approaches lead to similar results, it is not true in the pendulum case. This discrepancy can be traced back to the greater nonlinearity of the pendulum's ODE system compared to the oscillator one. This translates into a greater sensitivity to the control input and makes the training in-the-loop a complex local optimization problem. It was observed that the NN's weight initialization plays a crucial role in this. In fact, by varying the initialization, the results vary significantly. Few initialization approaches were tested, but none led to satisfactory results.

The training in-the-loop required lowering the integration step from 0.01 to 0.005 to stabilize the ODE propagation, which is another proof of the greater instability of the pendulum's ODE system and its sensitivity to the control input. On the other hand, OPGD-IGP can successfully find a good model because, during the evolutionary process, it learns to discard those solutions that lead to a failure of the ODE system's propagation. The results of the NN trained in-the-loop could improve by increasing its complexity and performing a thorough study of several initialization techniques. However, this would result in a non-interpretable model straying from the scope of this work.

6. Conclusions

This work applies OPGD-IGP, an IGP algorithm enhanced with a gradient-based LS strategy proposed by some of the authors in a previous work, for automatically designing a control law for a desired plant.

OPGD was designed for dealing with regression problems and leverages the backpropagation technique to evaluate the gradient of the objective function w.r.t the GP parameters. The backpropagation is impractical to use in control problems due to the implicit dependency of the state variables on the control variables. To overcome this issue, this study used the adjoint state method. The adjoint state method is a powerful mathematical approach that allows the evaluation of the gradient of an optimization problem involving a dynamical system with minimal computational effort and numerical errors compared to other techniques.

The proposed method was tested on two test cases: a harmonic oscillator controlled by a PD control law and an inverted pendulum on a cart controlled by a LQR control law. The objective of the experiments was to test the OPGD-IGP's capability to automatically design a control law similar, in terms of parameters and shape, to the reference one by leveraging the intra-evolution LS optimization. To understand the importance of the LS applied to GP, the performances achieved by OPGD-IGP have been compared with the ones achieved by IGP (a GP variant that does not involve any LS step), a Standard GP (SGP) without any LS step, and a feedforward NN. The NN was trained with two different approaches. First, it was trained on the data produced using the reference control laws. This training was performed to find the minimal NN topology necessary to capture the optimal control law behavior. Secondly, the NN with the minimal topology was trained in-the-loop, i.e., by interacting with the dynamical system as done by OPGD-IGP. The NN trained with this last approach is the one to consider when comparing NN with OPGD-IGP.

IGP and OPGD-IGP proved capable of performing the desired task, being able to produce a well-behaving control law for all the performed simulations with a good resemblance of shape and parameters with the reference control law. In particular, in the oscillator problem, IGP evolved 20/30 control laws with the same shape as the reference and similar parameters. On the other hand, OPGD-IGP achieved the desired shape and parameters in 30/30 simulations. Regarding the pendulum, IGP produced the desired shape and parameters only in 1/30 simulation while OPGD-IGP did it in 21/30 simulations.

Different performances were observed for SGP and the NN trained in-the-loop. Both performed well when applied to the oscillator test case. SGP produced a control law with the desired shape on 28/30 simulations. However, despite obtaining more models than IGP with the same shape as the reference, the resulting behaviors were more varied and less consistent than those produced by IGP. The NN trained in-the-loop was capable of controlling the system successfully, resulting in an objective function comparable to the one achieved by the GP-based algorithms. On the other hand, both SGP and NN trained in-the-loop showed poor performance when applied to the pendulum test case. This can be explained by the greater nonlinearity of the considered system, resulting in a more complex optimization problem that appeared extremely sensitive to the provided initial conditions.

These results confirm that GP is a valid alternative to classical approaches for automatically designing a control law. In particular, the use of LS combined with the GP evolutionary process led to inferring the optimal shape and parameters of the desired control law, in contrast with a GP approach not enhanced with an LS, where the control laws are different from each other and also different from the ground-truth. Moreover, comparing OPGD-IGP and SGP results on the oscillator case, it can be seen how the SGP can achieve the desired shape almost as often as the OGPD-IGP, although the parameters' values are randomly assigned. On the other hand, using an LS within the evolutionary process allows GP to find both the optimal shape and parameters. Finally, OPGD-IGP showed better performance than a feedforward NN. This result can be explained by the ability of GP to evolve models with different genotypes but with a phenotype close to the reference control law. Thus, GP can compensate for the sensitivity to the initial conditions in the pendulum test case by discarding those models that lead to a failure of the dynamical system propagation.

The obtained results have important implications, such as allowing control practitioners to automate the control law design process and explore new control law formulations when dealing with complex nonlinear problems. In fact, the results show that an optimal control law can be produced automatically also by considering the full nonlinear system.

Future research will focus on four directions. First, it would be interesting to apply OPGD-IGP online to create an Intelligent Control (IC) system. This would fully exploit the LS phase to adapt to unforeseen disturbances. Second, OPGD-IGP could be applied to systems with greater nonlinearities to automatically develop control schemes that otherwise would require an extensive design effort from the engineers. Third, the comparison between IGP and OPGD-IGP on the oscillator case shed light on the benefits of promoting exploration during the evolutionary process. It would be interesting to analyze the effects of a randomized initialization of the learnable parameters during the evolutionary process. This approach could lead to the exploitation of different local minima through the LS and allow the discovery of novel and better-performing control schemes. Lastly, a comparison with other AI-based approaches to generate interpretable control models should be performed. Control policies generated by GP in an RL framework have exhibited promising performance in similar tasks. A comparison with this approach could shed light on the advantages and limitations of the two learning methods. Such a comparison may also provide deeper insight into the poor performance of the NN trained in the loop, as discussed in this work. This observed behavior contrasts with other works in existing literature, where NNs trained in an RL framework show good performance across diverse domains.

CRediT authorship contribution statement

Francesco Marchetti: Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Investigation, Conceptualization. **Gloria Pietropolli:** Writing – original draft, Resources,

Methodology. **Federico Julian Camerota Verdù:** Writing – review & editing, Writing – original draft, Software, Resources, Methodology. **Mauro Castelli:** Writing – review & editing, Supervision, Conceptualization. **Edmondo Minisci:** Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work was supported by national funds through the FCT (Fundação para a Ciência e a Tecnologia), Portugal by the project UIDB/04152/2020-Centro de Investigação em Gestão de Informação – MagIC/NOVA IMS; and by the SPECIES Society through the SPECIES Scholarship 2022.

Appendix A. Neural networks training from data: settings and results

This appendix contains the settings used to train the NNs from the data and a summary of the training outcome.

A.1. Dataset

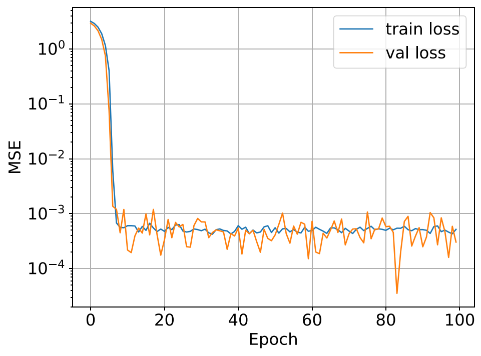
10000 samples were generated using a Latin Hypercube Sampling between [−2,2] for all the input features. These were then passed to Eqs. (16) and (18) to generate the corresponding output data for the oscillator and pendulum test cases. This way, one dataset for the oscillator case and one dataset for the pendulum case were created. The datasets were then split into train+validation (80%) and test datasets (20%). The train+validation dataset was further split into train (80%) and validation (20%) datasets.

A.2. Architecture and settings

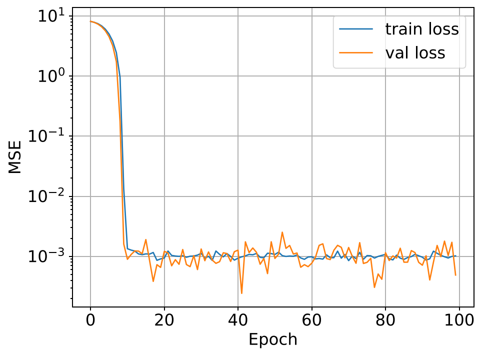
For both test cases, a minimal architecture consisting of one hidden layer with one neuron was used. This architecture proved sufficient to learn the optimal control laws from the data, as reported in Appendix A.3. Linear activation functions were used for each layer since the target model was a linear one. The weights were initialized with the Glorot uniform initialization, and the biases were initialized as zero. Considering all this, the NN model for the oscillator test case contains five tunable parameters, while the one used in the pendulum case contains seven parameters. The difference lies in the different number of inputs.

A.3. Training

The training was performed with the Adam optimizer with a learning rate of 0.001 for 100 epochs. The MSE was used as a loss function. The plots of the training and validation losses are depicted in Fig. A.15,

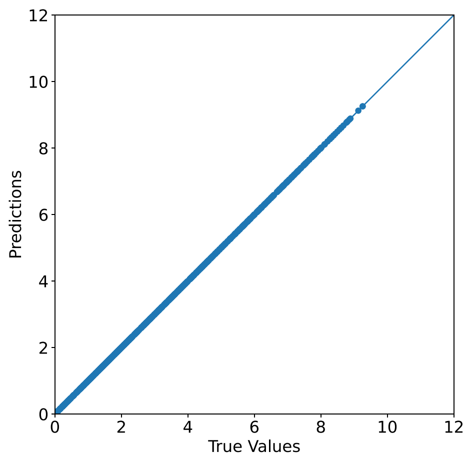


(a) Oscillator

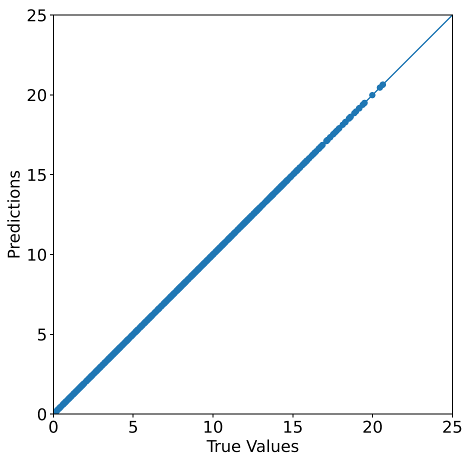


(b) Pendulum

Fig. A.15. Train and validation losses for the oscillator and pendulum test cases.



(a) Oscillator



(b) Pendulum

Fig. A.16. Comparison of true and predicted output using the test dataset.

while the prediction performances on the test data are depicted in Fig. A.16.

The models obtained are listed below and can be compared with Eqs. (16) and (17).

$$\begin{aligned} u_{NN,Data_{oscillator}} &= -1.966(0.891e_x + 1.530e_v + 0.297) + 0.585 = \\ &= -1.752e_x - 3.009e_v + 0.000262 \end{aligned}$$

$$\begin{aligned} u_{NN,Data_{pendulum}} &= -3.133(-0.319e_x - 0.452e_v + 2.594e_\theta + 0.390e_\omega + \\ &\quad + 0.201) + 0.631 = \\ &= -1.000e_x + 1.418e_v - 8.131e_\theta - 1.222e_\omega - 0.000419 \end{aligned}$$

Appendix B. Produced control laws

This appendix contains the models produced in all the simulations performed with SGP, IGP, and OPGD-IGP. The reported models are obtained by algebraically simplifying the models produced by the GP algorithms.

B.1. Oscillator

B.1.1. SGP

$$\begin{aligned}
u_{SGP_1} &= -1.437e_x - 2.874e_v \\
u_{SGP_2} &= -1.418e_x - 2.836e_v \\
u_{SGP_3} &= -1.456e_x - 2.912e_v \\
u_{SGP_4} &= -1.421e_x - 2.842e_v - 0.011 \\
u_{SGP_5} &= -2.118e_x - 3.566e_v \\
u_{SGP_6} &= -1.421e_x - 2.842e_v \\
u_{SGP_7} &= -1.413e_x - 2.826e_v \\
u_{SGP_8} &= -1.43e_x - 2.86e_v \\
u_{SGP_9} &= -1.424e_x - 2.848e_v \\
u_{SGP_{10}} &= -1.963e_x - 3.309e_v \\
u_{SGP_{11}} &= -1.407e_x - 2.814e_v \\
u_{SGP_{12}} &= -1.406e_x - 2.751e_v \\
u_{SGP_{13}} &= -1.421e_x - 2.694e_v \\
u_{SGP_{14}} &= -1.674e_x - 2.824e_v \\
u_{SGP_{15}} &= -2e_x - 3.42e_v \\
u_{SGP_{16}} &= -1.418e_x - 2.836e_v \\
u_{SGP_{17}} &= -1.422e_x - 2.844e_v \\
u_{SGP_{18}} &= -1.464e_x - 2.928e_v \\
u_{SGP_{19}} &= -1.506e_x - 2.819e_v \\
u_{SGP_{20}} &= -1.429e_x - 2.858e_v \\
u_{SGP_{21}} &= -1.835e_x - 3.115e_v \\
u_{SGP_{22}} &= -1.806e_x - 2.806e_v \\
u_{SGP_{23}} &= -1.67e_x - 3.082e_v \\
u_{SGP_{24}} &= -1.743e_x - 2.743e_v \\
u_{SGP_{25}} &= -1.421e_x - 2.711e_v \\
u_{SGP_{26}} &= -1.481e_x - 2.718e_v \\
u_{SGP_{27}} &= -1.426e_x - 2.852e_v \\
u_{SGP_{28}} &= -2.097e_x - 3.368e_v \\
u_{SGP_{29}} &= -1.457e_x - 2.914e_v - 0.0393 \\
u_{SGP_{30}} &= -1.419e_x - 2.838e_v
\end{aligned}$$

B.1.2. IGP

$$\begin{aligned}
u_{IGP_1} &= -1.775e_x - 3.059e_v \\
u_{IGP_2} &= -1.868e_x - 3.181e_v \\
u_{IGP_3} &= -1.831e_x - 3.123e_v \\
u_{IGP_4} &= -2e_x - 3.324e_v \\
u_{IGP_5} &= e_x(0.378e_v - 1.307) - 3.324e_v \\
u_{IGP_6} &= -1.853e_x - 3.157e_v \\
u_{IGP_7} &= -1.823e_x - 3.142e_v \\
u_{IGP_8} &= -1.771e_x - 3.108e_v \\
u_{IGP_9} &= -1.875e_x - 3.178e_v \\
u_{IGP_{10}} &= -1.912e_x - 3.234e_v \\
u_{IGP_{11}} &= -2e_x + 0.115(-e_v - 0.208)e_v - 3.517e_v \\
u_{IGP_{12}} &= -1.614e_x + 1.614(0.086e_v - 0.0265)e_x - 3.228e_v \\
u_{IGP_{13}} &= -1.841e_x - 3.138e_v \\
u_{IGP_{14}} &= -1.848e_x - 3.094e_v \\
u_{IGP_{15}} &= -1.871e_x - 3.159e_v \\
u_{IGP_{16}} &= -2e_x - 3.68e_v - 0.139e_v^2 \\
u_{IGP_{17}} &= -1.805e_x - (0.788e_v + 2.840)e_v - 1.805e_v \\
u_{IGP_{18}} &= -1.936e_x - 3.281e_v \\
u_{IGP_{19}} &= (e_v - 0.34(e_v + e_x)^2)(e_x - 3.205) \\
u_{IGP_{20}} &= -1.986e_x - 3.301e_v \\
u_{IGP_{21}} &= -e_x - 2e_v + 0.655(e_v + e_x)(e_x - 4.639) \\
u_{IGP_{22}} &= -1.965e_x - 3.335e_v \\
u_{IGP_{23}} &= -2.384e_x - 3.462e_v - 2.384e_x(-0.093e_v - 0.304) \\
u_{IGP_{24}} &= -1.881e_x - 3.183e_v \\
u_{IGP_{25}} &= -1.841e_x - 3.198e_v \\
u_{IGP_{26}} &= -1.829e_x - 3.134e_v \\
u_{IGP_{27}} &= 1.749e_v(0.00499e_x^2 - 1.749) - 1.749e_x \\
u_{IGP_{28}} &= -1.859e_x - 3.187e_v \\
u_{IGP_{29}} &= -1.892e_x - 3.191e_v \\
u_{IGP_{30}} &= -1.448e_x - 4.804e_v + 0.465e_v(e_x + 2.493)
\end{aligned}$$

B.1.3. OPGD-IGP

$$\begin{aligned}
u_{OPGD-IGP_1} &= -1.854e_x - 3.158e_v \\
u_{OPGD-IGP_2} &= -1.854e_x - 3.158e_v \\
u_{OPGD-IGP_3} &= -1.854e_x - 3.158e_v \\
u_{OPGD-IGP_4} &= -1.854e_x - 3.158e_v \\
u_{OPGD-IGP_5} &= -1.854e_x - 3.158e_v \\
u_{OPGD-IGP_6} &= -1.854e_x - 3.158e_v \\
u_{OPGD-IGP_7} &= -1.854e_x - 3.158e_v \\
u_{OPGD-IGP_8} &= -1.854e_x - 3.158e_v \\
u_{OPGD-IGP_9} &= -1.854e_x - 3.158e_v \\
u_{OPGD-IGP_{10}} &= -1.854e_x - 3.158e_v \\
u_{OPGD-IGP_{11}} &= -1.854e_x - 3.158e_v \\
u_{OPGD-IGP_{12}} &= -1.854e_x - 3.158e_v \\
u_{OPGD-IGP_{13}} &= -1.854e_x - 3.158e_v \\
u_{OPGD-IGP_{14}} &= -1.854e_x - 3.158e_v \\
u_{OPGD-IGP_{15}} &= -1.854e_x - 3.158e_v
\end{aligned}$$

$u_{OPGD-IGP_{16}} = -1.854e_x - 3.158e_v$
$u_{OPGD-IGP_{17}} = -1.854e_x - 3.158e_v$
$u_{OPGD-IGP_{18}} = -1.854e_x - 3.158e_v$
$u_{OPGD-IGP_{19}} = -1.854e_x - 3.158e_v$
$u_{OPGD-IGP_{20}} = -1.854e_x - 3.158e_v$
$u_{OPGD-IGP_{21}} = -1.854e_x - 3.158e_v$
$u_{OPGD-IGP_{22}} = -1.854e_x - 3.158e_v$
$u_{OPGD-IGP_{23}} = -1.854e_x - 3.158e_v$
$u_{OPGD-IGP_{24}} = -1.854e_x - 3.158e_v$
$u_{OPGD-IGP_{25}} = -1.854e_x - 3.158e_v$
$u_{OPGD-IGP_{26}} = -1.854e_x - 3.158e_v$
$u_{OPGD-IGP_{27}} = -1.854e_x - 3.158e_v$
$u_{OPGD-IGP_{28}} = -1.854e_x - 3.158e_v$
$u_{OPGD-IGP_{29}} = -1.854e_x - 3.158e_v$
$u_{OPGD-IGP_{30}} = -1.854e_x - 3.158e_v$

B.2. Pendulum

B.2.1. SGP

$u_{SGP_1} = e_\omega + e_\theta e_x(6.557 - e_\omega) + e_x(e_\omega e_v + e_\omega - e_v)$
$u_{SGP_2} = -e_\omega^2 - e_\omega - 2e_\theta + e_v$
$u_{SGP_3} = -e_\omega - 3.317e_\theta(e_\omega + e_v e_x + 2.495) + e_v + 0.546e_x$
$u_{SGP_4} = -e_\omega^2 - e_\omega - 2e_\theta + e_v$
$u_{SGP_5} = e_\theta(-0.48e_\omega e_\theta(2.085e_\omega - e_v + 7.325) - 0.56e_\theta + 2e_v - 2.527)$
$u_{SGP_6} = -e_\omega^2 + e_v + (e_\omega + 2e_\theta)(e_\theta + e_x)$
$u_{SGP_7} = -e_\omega - 4.05e_\theta + 2.05e_v + e_x$
$u_{SGP_8} = e_\theta(e_\omega - 4.203)(e_v + 5.389) - e_v$
$u_{SGP_9} = e_\theta(e_\omega + e_v + e_x - 13.233) - e_v$
$u_{SGP_{10}} = -e_\omega - 7.035e_\theta + 2e_v + e_x$
$u_{SGP_{11}} = -e_\omega - 3e_\theta + e_v - 0.386$
$u_{SGP_{12}} = -1.022e_\omega - e_\theta + e_v - 0.204$
$u_{SGP_{13}} = -e_\omega - e_\theta(8.154 - e_\omega) + e_v^2 + e_v + e_x$
$u_{SGP_{14}} = -0.919e_\omega + 0.919e_\theta(e_\omega - 6.911) + 1.838e_v + 0.919e_x$
$u_{SGP_{15}} = -e_\omega - 3e_\theta + 3e_v$
$u_{SGP_{16}} = -e_\omega e_v^2 - e_\omega - 7.264e_\theta + e_v(e_v - 0.066) + e_v + e_x$
$u_{SGP_{17}} = -e_\omega - 2e_\theta + e_v - 0.046e_x - 0.428$
$u_{SGP_{18}} = -e_\omega - 17.61e_\theta + e_v^2 + e_x$
$u_{SGP_{19}} = e_\omega e_v(0.687e_\theta e_x - 0.687e_v + 0.687e_x + 1.025) - 3.209e_\theta$
$u_{SGP_{20}} = -e_\omega + e_\theta(e_\omega - 5.807) + 2e_v + e_x$
$u_{SGP_{21}} = -e_\theta - (-e_\omega + 2e_v)(e_\theta + e_x) + (-5.723e_\theta + e_v)(e_v - e_x)$
$u_{SGP_{22}} = -0.089e_\omega - 5.937e_\theta - e_x^2 + 3.986$
$u_{SGP_{23}} = -2.803e_\omega + e_\theta - 1.803e_v + (-e_\theta + e_v)(-8.89e_v - 8.89e_x) - 1.015$
$u_{SGP_{24}} = -e_\omega - 7.453e_\theta + 2e_v + e_x$
$u_{SGP_{25}} = -e_\omega - 5.501e_\theta^2 + e_\theta(e_\omega + e_x) + 2.935e_v$
$u_{SGP_{26}} = (0.023e_\omega + e_\theta)(4.033e_\omega - 2e_\theta - e_v + e_x + 0.259)$
$u_{SGP_{27}} = -e_\omega - 5e_\theta + 2e_v + e_x$
$u_{SGP_{28}} = -e_\omega + 51.050e_\theta(1.045e_v - 1.254)$
$u_{SGP_{29}} = -2e_\omega e_v - e_\omega - 2e_\theta - e_v^2 + e_v$
$u_{SGP_{30}} = -e_\omega - 2.926e_\theta(-7.445e_\omega e_\theta e_x - e_\theta e_x + 2.791) + 2e_v + e_x$

B.2.2. IGP

$$\begin{aligned}
u_{IGP_1} =& - 1.280e_\omega - 8.095e_\theta + 1.560e_v + e_x + e_\theta(e_\omega + e_\theta - e_v - e_x)\\
u_{IGP_2} =& - e_\omega - e_\theta(-9.963e_\omega e_\theta(e_\omega - e_v(e_\theta + 1.5)) + 6.686) - e_\theta +\\
&+ e_v(e_\theta + 1.5) + e_x\\
u_{IGP_3} =& 9.268e_\omega e_\theta^2 - e_\omega - 9.555e_\theta + e_v^2 - e_v(-e_x - 1.686) + e_x\\
u_{IGP_4} =& - e_\omega + e_v + (-6.765e_\theta + e_x)(e_\theta + 0.829)(e_\theta^2(e_\omega +\\
&- 7.736)(e_\omega + e_v) + 0.829)\\
u_{IGP_5} =& - 1.155e_\omega - 6.993e_\theta + 1.47e_v + e_x\\
u_{IGP_6} =& - e_\omega - e_\theta(8.693e_\omega e_\theta(-e_\omega + e_v) - 2.428e_\theta + 6.667) + 1.428e_v + e_x\\
u_{IGP_7} =& - e_\omega + e_\theta(e_v e_x + e_x - 6.787) + e_v(e_v + e_x + 0.664) + e_v + e_x\\
u_{IGP_8} =& - 1.16e_\omega + e_\theta(-1.677e_\omega e_v + 5.488e_\theta - 6.197) + 1.345e_v + e_x\\
u_{IGP_9} =& - e_\omega - e_\theta(9.96e_\omega e_\theta - 2e_\theta + e_v - e_x + 4.5) + 1.388e_v + e_x\\
u_{IGP_{10}} =& - e_\omega - 7.709e_\theta + e_v + e_x + (e_x + 2.071)(e_\theta(e_\theta - 6.909) +\\
&+ e_v)(-e_\theta + e_x + 2.071)\\
u_{IGP_{11}} =& - e_\omega - e_\theta(e_\omega + 9.063) + e_v(-e_\theta + e_v + e_x) + 1.715716e_v + e_x\\
u_{IGP_{12}} =& - e_\omega - e_\theta(-e_\omega - 7.203e_\theta + e_v + e_x + 8.802) + e_\theta + e_v(e_\theta + 0.444) +\\
&+ e_v + e_x\\
u_{IGP_{13}} =& (e_\theta + 0.031e_v((e_\omega - 2e_\theta)(e_v + 1.835) - 1.159))(e_\theta e_x - 1.835e_\theta +\\
&- 2.755)\\
u_{IGP_{14}} =& - e_\omega + e_\theta(0.439e_\omega + 3e_\theta - 5.753) - e_\theta + 1.418e_v + e_x\\
u_{IGP_{15}} =& - e_\omega + e_\theta(e_\omega + e_v e_x - 6.116) - e_\theta + e_v + e_x - (e_\omega +\\
&- 1.57e_v)(e_\theta + 0.254)\\
u_{IGP_{16}} =& - e_\omega - 8.478e_\theta + e_v(-2e_\theta + e_x) + e_v(e_v - 0.419) + 2e_v + e_x\\
u_{IGP_{17}} =& - e_\omega - e_\theta + 2e_v + e_x + (0.411 - e_\theta)(-0.34e_\omega - 0.660e_\theta - e_v +\\
&+ 6.702) - 2.78\\
u_{IGP_{18}} =& - e_\omega + e_\theta(e_\omega - e_\theta(-2e_\omega^2 - e_x - 9.545) - 2e_\theta - 6.259) +\\
&+ 1.377e_v + e_x\\
u_{IGP_{19}} =& - 1.113e_\omega - 7.299e_\theta - 0.113e_v(-e_\omega - e_\theta - e_x + 3.652) + 2e_v + e_x\\
u_{IGP_{20}} =& - e_\omega + 1.105e_\theta^2 - 0.187e_\theta e_v - 5.726e_\theta + 1.187e_v + 0.813e_x\\
u_{IGP_{21}} =& - e_\omega - 5.08e_\theta + e_v - 0.09072e_x(e_\omega - e_\theta) + 0.676e_x\\
u_{IGP_{22}} =& - e_\omega - e_\theta(-3e_\theta - e_v + 6.92) - e_v(-0.295e_\omega - 0.346) + e_v + e_x\\
u_{IGP_{23}} =& - 1.266e_\omega + e_\theta(e_\omega - e_\theta - e_v - 6.694) + 1.532e_v + e_x\\
u_{IGP_{24}} =& - e_\omega - 8.525e_\theta + e_v(-e_\omega e_v - 3e_\theta + e_v^2 + e_x) + 2e_v + e_x\\
u_{IGP_{25}} =& - e_\omega + e_\theta(e_\omega e_\theta(4.593e_\omega - 6.515) + e_\theta - 6.515) - 0.452e_\theta +\\
&+ 1.452e_v + e_x\\
u_{IGP_{26}} =& - 0.0241e_\omega(e_v + e_x) - 0.916e_\omega - 5.507e_\theta + 1.083e_v + 0.711e_x\\
u_{IGP_{27}} =& - e_\omega - e_\theta(e_v(-e_\theta(e_\theta + 4.417) + 0.742) + 8.424) + e_v(-e_\theta +\\
&+ e_v + e_x + 0.674) + e_v + e_x\\
u_{IGP_{28}} =& - e_\omega + e_\theta(-e_\theta(e_\omega e_\theta + e_\omega)(-e_\omega e_x + e_\omega - 6.744) - 5.207) + e_v +\\
&+ 0.663e_x\\
u_{IGP_{29}} =& - 1.169e_\omega + e_\theta^2 - 6.976e_\theta + 1.413e_v + e_x + 0.0148\\
u_{IGP_{30}} =& - e_\omega - e_\theta(-e_\omega - 7.074e_\theta + 0.751e_v + e_x + 7.241) + 1.436e_v + e_x
\end{aligned}$$

B.2.3. OPGD-IGP

$$\begin{aligned}
u_{OPGD-IGP_1} =& - 0.951e_\omega - 0.0274e_\theta(0.0815e_\omega e_\theta + 0.121e_\theta) - 5.839e_\theta +\\
&+ 1.161e_v + 0.780e_x\\
u_{OPGD-IGP_2} =& - 0.951e_\omega - 5.839e_\theta + 1.160e_v + 0.780e_x\\
u_{OPGD-IGP_3} =& - 0.953e_\omega - 5.844e_\theta + 1.162e_v + 0.781e_x\\
u_{OPGD-IGP_4} =& - 0.952e_\omega - 5.840e_\theta + 1.161e_v + 0.780e_x
\end{aligned}$$

$$u_{OPGD-IGP_5} = -0.0172e_\omega e_x - 0.986e_\omega - 5.847e_\theta + 1.162e_v + 0.781e_x$$
$$u_{OPGD-IGP_6} = -0.952e_\omega - 5.841e_\theta + 1.161e_v + 0.781e_x$$
$$u_{OPGD-IGP_7} = -0.949e_\omega - 5.903e_\theta + 1.174e_v + \\ -0.00900e_x(2.347e_\theta - 1.269e_v) + 0.780e_x$$
$$u_{OPGD-IGP_8} = -0.952e_\omega - 5.843e_\theta + 1.162e_v + 0.781e_x$$
$$u_{OPGD-IGP_9} = -0.952e_\omega - 5.843e_\theta + 1.162e_v + 0.781e_x$$
$$u_{OPGD-IGP_{10}} = -0.952e_\omega - 5.842e_\theta + 1.161e_v + 0.781e_x$$
$$u_{OPGD-IGP_{11}} = -0.952e_\omega - 5.842e_\theta + 1.161e_v + 0.781e_x$$
$$u_{OPGD-IGP_{12}} = -0.952e_\omega - 5.842e_\theta + 1.161e_v + 0.781e_x$$
$$u_{OPGD-IGP_{13}} = -0.954e_\omega + 1.090e_\theta e_v(-1.030e_\omega e_\theta + 0.954e_\theta) + \\ -5.806e_\theta + 1.158e_v + 0.783e_x$$
$$u_{OPGD-IGP_{14}} = -0.952e_\omega - 5.843e_\theta + 1.162e_v + 0.781e_x$$
$$u_{OPGD-IGP_{15}} = -0.952e_\omega - 5.839e_\theta + 1.161e_v + 0.780e_x$$
$$u_{OPGD-IGP_{16}} = -0.953e_\omega - 0.010e_\theta e_x(0.998e_\theta - 0.994e_v) + \\ -5.835e_\theta + 1.162e_v + 0.782e_x$$
$$u_{OPGD-IGP_{17}} = -0.952e_\omega - 5.843e_\theta + 1.162e_v + 0.781e_x$$
$$u_{OPGD-IGP_{18}} = -0.952e_\omega - 5.844e_\theta + 1.162e_v + 0.781e_x$$
$$u_{OPGD-IGP_{19}} = -0.953e_\omega - 5.844e_\theta + 1.162e_v + 0.781e_x$$
$$u_{OPGD-IGP_{20}} = -0.952e_\omega + 0.0854e_\theta^2 - 5.834e_\theta + 1.161e_v + 0.781e_x$$
$$u_{OPGD-IGP_{21}} = -0.952e_\omega - 5.844e_\theta + 1.162e_v + 0.781e_x$$
$$u_{OPGD-IGP_{22}} = -0.951e_\omega + 0.00449e_\theta(0.999e_\omega - 1.999e_\theta - 0.999e_v) + \\ -5.835e_\theta + 1.160e_v + 0.780e_x$$
$$u_{OPGD-IGP_{23}} = -0.952e_\omega - 5.843e_\theta + 1.161e_v + 0.781e_x$$
$$u_{OPGD-IGP_{24}} = -0.952e_\omega - 5.842e_\theta + 1.161e_v + 0.781e_x$$
$$u_{OPGD-IGP_{25}} = -0.952e_\omega - 5.842e_\theta + 1.161e_v + 0.781e_x$$
$$u_{OPGD-IGP_{26}} = -0.952e_\omega - 5.843e_\theta + 1.162e_v + 0.781e_x$$
$$u_{OPGD-IGP_{27}} = -0.952e_\omega - 5.842e_\theta + 1.161e_v + 0.781e_x$$
$$u_{OPGD-IGP_{28}} = -0.950e_\omega - 5.862e_\theta + 1.153e_v - 0.00907e_x^2 + 0.7631e_x$$
$$u_{OPGD-IGP_{29}} = -0.952e_\omega - 5.840e_\theta + 1.161e_v + 0.780e_x$$
$$u_{OPGD-IGP_{30}} = -0.952e_\omega + 0.0264e_\theta(0.999e_\theta - 0.999e_v(1.000e_\omega - 0.999e_v)) \\ -5.852e_\theta + 1.162e_v + 0.782e_x$$

References

[1] J.R. Koza, Genetic programming as a means for programming computers by natural selection, Stat. Comput. 4 (2) (1994) 87–112.

[2] M. Castelli, L. Trujillo, L. Vanneschi, S. Silva, E. Z-Flores, P. Legrand, Geometric semantic genetic programming with local search, in: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, 2015, pp. 999–1006.

[3] L. Muñoz, L. Trujillo, S. Silva, M. Castelli, L. Vanneschi, Evolving multidimensional transformations for symbolic regression with M3GP, Memet. Comput. 11 (2) (2019) 111–126.

[4] G. Pietropolli, L. Manzoni, A. Paoletti, M. Castelli, Combining geometric semantic gp with gradient-descent optimization, in: European Conference on Genetic Programming (Part of EvoStar), Springer, 2022, pp. 19–33.

[5] G. Pietropolli, F.J. Camerota Verdù, L. Manzoni, M. Castelli, Parametrizing GP trees for better symbolic regression performance through gradient descent., in: Proceedings of the Companion Conference on Genetic and Evolutionary Computation, 2023, pp. 619–622.

[6] M. Zhang, W. Smart, Genetic programming with gradient descent search for multiclass object classification, in: European Conference on Genetic Programming, Springer, 2004, pp. 399–408.

[7] F. Marchetti, E. Minisci, A. Riccardi, Towards Intelligent Control via Genetic Programming, in: Proceedings of the International Joint Conference on Neural Networks, 2020, http://dx.doi.org/10.1109/IJCNN48605.2020.9207694.

[8] C.-H. Chiang, A genetic programming based rule generation approach for intelligent control systems, in: 2010 International Symposium on Computer, Communication, Control and Automation (3CA), vol. 1, IEEE, 2010, pp. 104–107.

[9] K.J. Åström, R.M. Murray, Feedback Systems, Feedback Systems - An Introduction for Scientists and Engineers, Princeton University Press, 2010, p. 545, http://dx.doi.org/10.2307/j.ctvcm4gdk, URL http://www.jstor.org/stable/10.2307/j.ctvcm4gdk.

[10] C. Utama, B. Karg, C. Meske, S. Lucia, Explainable artificial intelligence for deep learning-based model predictive controllers, in: 2022 26th International Conference on System Theory, Control and Computing, ICSTCC, 2022, pp. 464–471, http://dx.doi.org/10.1109/ICSTCC55426.2022.9931794.

[11] C.K. Oh, G.J. Barlow, Autonomous controller design for unmanned aerial vehicles using multi-objective genetic programming, in: Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753), 2004, pp. 1538–1545 Vol.2, http://dx.doi.org/10.1109/CEC.2004.1331079.

[12] A. Bourmistrova, S. Khantsis, Genetic Programming in Application to Flight Control System Design Optimisation, New Ach. Evol. Comput. (2010).

[13] R.T.Q. Chen, Y. Rubanova, J. Bettencourt, D. Duvenaud, Neural ordinary differential equations, in: Proceedings of the 32nd International Conference on Neural Information Processing Systems, NIPS '18, Curran Associates Inc., Red Hook, NY, USA, 2018, pp. 6572–6583.

[14] L.S. Pontryagin, E. Mishchenko, V. Boltyanskii, R. Gamkrelidze, The Mathematical Theory of Optimal Processes, 1962.

[15] F. Neri, C. Cotta, Memetic algorithms and memetic computing optimization: A literature review, Swarm Evol. Comput. 2 (2012) 1–14.

[16] X. Chen, Y.-S. Ong, M.-H. Lim, K.C. Tan, A multi-facet survey on memetic computation, IEEE Trans. Evol. Comput. 15 (5) (2011) 591–607.

[17] L. Trujillo, O. Schütze, P. Legrand, et al., Evaluating the effects of local search in genetic programming, in: EVOLVE-a Bridge Between Probability, Set Oriented Numerics, and Evolutionary Computation V, Springer, 2014, pp. 213–228.

[18] G. Pietropolli, L. Manzoni, A. Paoletti, M. Castelli, On the hybridization of geometric semantic GP with gradient-based optimizers, Genet. Programm. Evol. Mach. 24 (2) (2023) 1–20.

[19] B.E. Eskridge, D.F. Hougen, Imitating success: A memetic crossover operator for genetic programming, in: Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753), vol. 1, IEEE, 2004, pp. 809–815.

[20] P. Wang, K. Tang, E.P. Tsang, X. Yao, A memetic genetic programming with decision tree-based local search for classification problems, in: 2011 IEEE Congress of Evolutionary Computation, CEC, IEEE, 2011, pp. 917–924.

[21] A. Topchy, W.F. Punch, et al., Faster genetic programming based on local gradient search of numeric leaf values, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), vol. 155162, Morgan Kaufmann San Francisco, CA, 2001.

[22] G. Nadizar, F. Garrow, B. Sakallioglu, L. Canonne, S. Silva, L. Vanneschi, An investigation of geometric semantic GP with linear scaling, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2023, pp. 1165–1174.

[23] M. Graff, R. Pena, A. Medina, Wind speed forecasting using genetic programming, in: 2013 IEEE Congress on Evolutionary Computation, IEEE, 2013, pp. 408–415.

[24] W. Smart, M. Zhang, Continuously evolving programs in genetic programming using gradient descent, in: Proceedings of the 7th Asia-Pacific Conference on Complex Systems, 2004.

[25] M. Kommenda, G. Kronberger, S. Winkler, M. Affenzeller, S. Wagner, Effects of constant optimization by nonlinear least squares minimization in symbolic regression, in: Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation, 2013, pp. 1121–1128.

[26] J. Harrison, M. Virgolin, T. Alderliesten, P. Bosman, Mini-batching, gradient-clipping, first-versus second-order: What works in gradient-based coefficient optimisation for symbolic regression? in: Proceedings of the Genetic and Evolutionary Computation Conference, 2023, pp. 1127–1136.

[27] F. Marchetti, E. Minisci, Inclusive Genetic Programming, in: T. Hu, N. Lourenço, E. Medvet (Eds.), in: Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 12691 LNCS, Springer International Publishing, Cham, 2021, pp. 51–65, http://dx.doi.org/10.1007/978-3-030-72812-0_4.

[28] J. Koza, M. Keane, J. Yu, F. Bennett, III, W. Mydlowec, Automatic Creation of Human-Competitive Programs and Controllers by Means of Genetic Programming, Genet. Programm. Evol. Mach. 1 (1/2) (2000) 121–164, http://dx.doi.org/10.1023/A:1010076532029.

[29] C.F. Verdier, M. Mazo, Jr., Formal Controller Synthesis via Genetic Programming, IFAC-PapersOnLine 50 (1) (2017) 7205–7210, http://dx.doi.org/10.1016/j.ifacol.2017.08.1362, URL http://www.sciencedirect.com/science/article/pii/S2405896317318979.

[30] K. Łapa, K. Cpałka, A. Przybył, Genetic programming algorithm for designing of control systems, Inf. Technol. Control 47 (4) (2018) 668–683, http://dx.doi.org/10.5755/j01.itc.47.4.20795.

[31] K. Danai, W.G. La Cava, Controller design by symbolic regression, Mech. Syst. Signal Process. 151 (2021) 107348, http://dx.doi.org/10.1016/j.ymssp.2020.107348.

[32] G.W. Irwin, K. Warwick, K.J. Hunt, Neural Network Applications in Control, 1995, p. 312.

[33] S.A. Emami, P. Castaldi, A. Banazadeh, Neural network-based flight control systems: Present and future, Annu. Rev. Control 53 (May) (2022) 97–137, http://dx.doi.org/10.1016/j.arcontrol.2022.04.006, arXiv:2206.05596.

[34] L. Böttcher, N. Antulov-Fantulin, T. Asikis, AI Pontryagin or how artificial neural networks learn to control dynamical systems, Nature Commun. 13 (1) (2022) 1–9, http://dx.doi.org/10.1038/s41467-021-27590-0.

[35] D. Hein, S. Udluft, T.A. Runkler, Generating interpretable fuzzy controllers using particle swarm optimization and genetic programming, in: GECCO 2018 Companion - Proceedings of the 2018 Genetic and Evolutionary Computation Conference Companion, 2018, pp. 1268–1275, http://dx.doi.org/10.1145/3205651.3208277, arXiv:1804.10960.

[36] D. Hein, S. Udluft, T.A. Runkler, Interpretable policies for reinforcement learning by genetic programming, Eng. Appl. Artif. Intell. 76 (April) (2018) 158–169, http://dx.doi.org/10.1016/j.engappai.2018.09.007, arXiv:1712.04170.

[37] D. Hein, S. Limmer, T.A. Runkler, Interpretable control by reinforcement learning, IFAC-PapersOnLine 53 (2) (2020) 8082–8089, http://dx.doi.org/10.1016/j.ifacol.2020.12.2277, arXiv:2007.09964.

[38] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.

[39] J. Nocedal, S.J. Wright, Numerical Optimization, in: Physical Review, second ed., Springer, NY, 2006.

[40] H.J. Kelley, Gradient theory of optimal flight paths, Ars J. 30 (10) (1960) 947–954.

[41] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, 2016.

[42] B.M. Bell, J.V. Burke, Algorithmic Differentiation of Implicit Functions and Optimal Values, in: Lecture Notes in Computational Science and Engineering, vol. 64 LNCSE, 2008, pp. 67–77, http://dx.doi.org/10.1007/978-3-540-68942-3_7, URL http://link.springer.com/10.1007/978-3-540-68942-3{_}7.

[43] C.C. Margossian, M. Betancourt, Efficient Automatic Differentiation of Implicit Functions, 2021, URL http://arxiv.org/abs/2112.14217, arXiv:2112.14217.

[44] Y. Ma, V. Dixit, M.J. Innes, X. Guo, C. Rackauckas, A Comparison of Automatic Differentiation and Continuous Sensitivity Analysis for Derivatives of Differential Equation Solutions, in: 2021 IEEE High Performance Extreme Computing Conference, HPEC 2021, (2) IEEE, 2021, pp. 1–9, http://dx.doi.org/10.1109/HPEC49654.2021.9622796, arXiv:1812.01892.

[45] A. Güne, G. Baydin, B.A. Pearlmutter, J.M. Siskind, Automatic Differentiation in Machine Learning: A Survey, J. Mach. Learn. Res. 18 (2018) 1–43.

[46] C.M. Pappalardo, D. Guida, Use of the adjoint method for controlling the mechanical vibrations of nonlinear systems, Machines 6 (2) (2018) http://dx.doi.org/10.3390/machines6020019.

[47] S.L. Brunton, J.N. Kutz, Data-Drive Science and Engineering: Machine Learning, Dynamical Systems and Control, vol. 53, (9) 2019.

[48] P. Virtanen, R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S.J. van der Walt, M. Brett, J. Wilson, K.J. Millman, N. Mayorov, A.R.J. Nelson, E. Jones, R. Kern, E. Larson, C.J. Carey, İ. Polat, Y. Feng, E.W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E.A. Quintero, C.R. Harris, A.M. Archibald, A.H. Ribeiro, F. Pedregosa, P. van Mulbregt, SciPy 1.0 Contributors, SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python, Nature Methods 17 (2020) 261–272, http://dx.doi.org/10.1038/s41592-019-0686-2.