xMLC - A Toolkit for Machine Learning Control

Guy Y. Cornejo Maceda, Francois Lusseyran, Bernd R. Noack

▶ **To cite this version:**

Guy Y. Cornejo Maceda, Francois Lusseyran, Bernd R. Noack. xMLC - A Toolkit for Machine Learning Control. Technische Universität Braunschweig, 2, 2022, Machine Learning Tools in Fluid Mechanics, 10.24355/dbbs.084-202208220937-0 . hal-04232473

HAL Id: hal-04232473
https://hal.science/hal-04232473v1

Submitted on 8 Oct 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

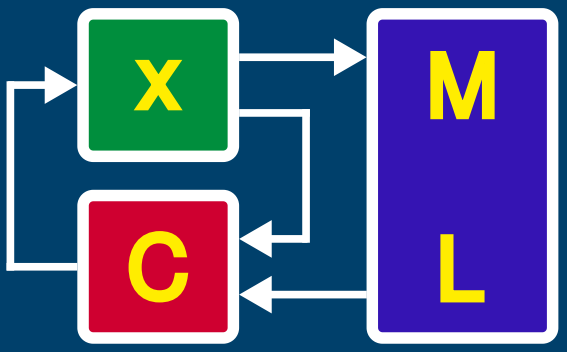
Distributed under a Creative Commons Attribution 4.0 International License

Machine Learning Tools in Fluid Mechanics, Vol 2

xMLC

A Toolkit for Machine Learning Control

First Edition



Guy Y. Cornejo Maceda,
François Lusseyran
& Bernd R. Noack

Shenzhen & Paris

Universitätsbibliothek
der Technischen Universität Braunschweig
2022

Machine Learning Tools in Fluid Mechanics

Series number: 2

xMLC

A Toolkit for Machine Learning Control

First Edition

Guy Y. Cornejo Maceda, François Lusseyran & Bernd R. Noack
Shenzhen & Paris

Series editors: Richard Semaan & Bernd R. Noack

Copyright © 2022 Guy Y. Cornejo Maceda, François Lusseyran & Bernd R. Noack
License: Creative Commons Attribution 4.0
DOI: 10.24355/dbbs.084-202208220937-0

Dr. Guy Y. Cornejo Maceda
Postdoctoral fellow
School of Mechanical Engineering
and Automation
Harbin Institute of Technology
Shenzhen Campus, Building L, Room 319
University Town, Xili
Shenzhen 518055, China
Email: Yoslan@hit.edu.cn

Prof. Dr. François Lusseyran
Laboratoire Interdisciplinaire
des Sciences du Numérique
Université Paris-Saclay, CNRS
Campus universitaire Bât. 507
Rue du Belvedère
F - 91405 Orsay, France
Email: Francois.Lusseyran@limsi.fr

Prof. Dr. Bernd R. Noack
National Talent Professor
School of Mechanical Engineering
and Automation
Harbin Institute of Technology
Shenzhen Campus, Building L, Room 2020
University Town, Xili
Shenzhen 518055, China
Email: Bernd.Noack@hit.edu.cn

Preface

xMLC is the second book of this '*Machine Learning Tools in Fluid Mechanics*' Series and focuses on Machine Learning Control (MLC). The objectives of this book are two-fold: First, provide an introduction to MLC for students, researchers, and newcomers on the field; and second, share an open-source code, `xMLC`, to automatically learn open- and closed-loop control laws directly in the plant with only a few executable commands.

This presented MLC algorithm is based on genetic programming and highlights the learning principles (exploration and exploitation). The need of balance between these two principles is illustrated with an extensive parametric study where the explorative and exploitative forces are gradually integrated in the optimization process. The provided software `xMLC` is an implementation of MLC. It builds on `OpenMLC` (Duriez et al., 2017) but replaces tree-based genetic programming but the linear genetic programming framework (Brameier and Banzhaf, 2006). The latter representation is preferred for its easier implementation of multiple-input multiple-output control laws and of the genetic operators (mutation and crossover). The handling of the software is facilitated by a step by step guide that shall help new practitioners to use the code within few minutes. We also provide detailed advice in using the code for other solvers and for experiments. The code is open-source and a GitHub version is available for future updates, options and add-ons.

`xMLC` is the result of collaborations of many students, engineers and researchers in Argentina, Canada, China, France, Germany, Japan, Jordan, Poland, Sweden, Spain, and USA since almost one decade. We deeply thank Marc Segond for initiating us into the world of genetic programming, Mattias Wahde for alerting us to the advantages of the linear version and Marc Schoenauer for advice regarding performance analysis and improvement. We are indebted to Thomas Duriez and Ruiying Li for sharing their own implementation of MLC and for their valuable advice. Special thanks are due to our numerous collaborators helping to improve MLC and to gain valuable experience in dozens of experiments and simulations: Markus Abel, Jean-Luc Aider, Jacques Borée, Jean-Paul Bonnet, Steven L. Brunton, Sylvain Caillou, Rodrigo Castellanos, Camila Chovet, Laurent Cordier, Christophe Cuvier, Antoine Debien, Joël Delville, Carine Fourment, Hiroaki Fukumoto, Ignacio de la Fuente, Nan Deng, Dewei Fan, Stefano Discetti, Jean Marc Foucault, Nan Gao, Nicolas Gautier, Fabien Harambat, Andrea Ianiro, Eurika Kaiser, Laurent Keirsbulck, Azeddine Kourta, Jean-Christophe Laurentie, Jean-Christophe Loiseau, Hao Li, Songqi Li, Yiqing Li, Congjun Liu, Yutong Liu, Jiayang Luo, Marc Lippert, Robert Martinuzzi, Lionel Mathelin, Nicolas Mazellier, Marek, Morzyński, Philipp Oswald, Akira Oyama, Vladimir Parezanovic, Pierre-Yves Passaggia, Luc Pastur, Cédric Raibaudo, Richard Semaan, Tamir Shaqarin, Ruixuan Shen, Hongke Shi, Michel Stanislas, Jianguo Tan, Kai, A. F. F. von Krbek and Eliott Varon.

Shenzhen and Paris in July 2022
Guy Y. Cornejo Maceda
François Lusseyran
Bernd R. Noack

Contents

Chapter 1

Introduction

This book shall facilitate a smooth application of machine learning control (Duriez et al., 2017) to numerical and experimental plants. Focus is placed on linear genetic programming control for nonlinear dynamics systems and, in particular, fluid flows.

Starting point of any control is the beautiful framework of linear theory (Doyle et al., 1992; Åström and Murray, 2010). Linear theory has been the key enabler for stabilization of fluid flow instabilities (Rowley and Williams, 2006; Kim and Bewley, 2007; Sipp et al., 2010) and can provide mathematically rigorous performance guarantees. Yet, linear dynamics has numerous implications which are uncommon for complex systems. For instance, the unforced system either converges to a single fixed point or diverges to infinity. Or, the frequency of periodic actuation does not affect other frequencies of the system. Linear methods can be extended to weakly nonlinear dynamics, e.g., via a nonlinear transformation to a linear system (input-output linearization) or via a meaningful local linearization (linear parameter-varying systems). Linear control theory is inapplicable for frequency crosstalk, i.e., if the actuation or system frequency effect other system frequencies. Yet, frequency crosstalk is a defining feature and often key enabler for turbulence control and many dynamical and complex systems.

Recently, machine learning has opened the new avenue of model-free nonlinear control. *Machine Learning Control (MLC)* can perform an automated control optimization of multiple-input (multiple-actuator) multiple-output (multiple-sensor) plants. MLC learns the control laws like mother nature—by clever trial and error. More specifically, a regression problem of the second kind is formulated in which the cost is minimized via the MIMO control law. Literature offers several approaches (Brunton and Noack, 2015; Brunton et al., 2020). The optimization of a parameterized control law, e.g., linear feedback, invites all methods of optimization, like Bayesian optimization (Blanchard et al., 2022), downhill simplex descent, Monte Carlo sampling, explorative gradient method (Li et al., 2022), genetic algorithm (Benard et al., 2016), particle swarm optimization (Shaqarin and Noack, 2022), and numerous other biologically inspired algorithms (Wahde, 2008). Cluster-based control infers the structure of the nonlinear control law from flow data and allows for downhill simplex optimization of simple feedbacks in only few dozen simulations or experiments (Nair et al., 2019).

A spectrum of optimizing general nonlinear laws has been invented in the 1990's. Neural network-based control (Lee et al., 1997) has been developed by the team of J. Kim for wall-turbulence skin friction reduction. Reinforcement learning (RL) (Sutton and Barto, 1998) has become popular in robotics and has been pioneered in flow control by J. Rabault's team (Rabault et al., 2019; Tang et al., 2020; Wang et al., 2022) and others (Fan et al., 2020). Genetic programming control (GPC) was discovered by Dracopoulos (1997) for stabilizing satellite motion and was later re-invented for turbulence control (Duriez et al., 2017) under the name *Machine Learning Control (MLC)*. First studies indicate that RL and GPC have overall similar performance with few application specific differentiations (Castellanos et al., 2022). Numerous efforts to accelerate and improve the performance of all approaches, e.g., gradient information has significantly reduced the learning time of GPC (Cornejo Maceda et al., 2021).

In the sequel, the term MLC will be narrowly used for GPC only. GPC is simple regression solver and seems the best tested version of MLC for turbulence control. In dozens of experiments and simulations, GPC has consistently outperformed other optimized control laws in their respective plant, often with unexpected new actuation mechanisms, e.g. frequency crosstalk mechanisms. GPC has even successfully learned a novel distributed actuation (Zhou et al., 2020). Successful applications include following:

1. Stabilization of a noisy linear oscillator (Duriez et al., 2017).
2. Chaotization of the forced Lorenz system (Duriez et al., 2014).

3. Stabilization of a generalized mean-field system (2 nonlinearly coupled oscillators as derived by Luchtenburg et al. (2009)) (Duriez et al., 2017)
4. Stabilization of a generalized mean-field system (3 nonlinearly coupled oscillators) (Li et al., 2018).
5. Stabilization of a 2D mixing layer in a direct numerical simulation (open loop) Li et al. (2020).
6. Energetization of a 2D mixing layer in a direct numerical simulation (open loop) Li et al. (2020).
7. Mixing layer energetization in a wind tunnel—feedback with 96 actuators in unison and 19 hot-wire sensors) (Parezanović et al., 2013; Parezanović et al., 2016).
8. Mixing increase behind a backward-facing step in a wind-tunnel experiment—20 actuators (unison), 3 sensors (Chovet et al., 2017).
9. Recirculation zone reduction behind a backward facing step in a water tunnel—feedback with single actuator, online PIV flow monitoring (Gautier et al., 2015).
10. Separation control of a turbulent boundary layer over a ramp—feedback with 54 active vortex generators (unison) and 2 sensors (Duriez et al., 2014; Debien et al., 2016).
11. Smart skin separation control —feedback with 2×30 actuators and 58 sensors [HIT 2022].
12. Lift-increase of a NACA0015 airfoil with an emulated plasma actuator in a direct numerical simulation [Joint work with H. Fukumoto & A. Oyama].
13. Lift-increase of a high-lift configuration at high Reynolds number in experiment—Open-loop with 33 actuators and 196 sensors El-Sayed et al. (2019)
14. Mitigation of dynamic bubble burst for stall suppression —feedback with 1 actuator and 1 sensor under transient conditions (Asai et al., 2021).
15. Open-loop stabilization of a fluidic pinball in a wind-tunnel experiment—open-loop with 3 actuators (Raibaudo et al., 2017).
16. Stabilization of a fluidic pinball in a simulation—feedback with 3 actuators and 9 sensors (Cornejo Maceda et al., 2021).
17. Ditto but with drag reduction (Cornejo Maceda et al., 2019).
18. Drag reduction of a D-shaped cylinder in a wind-tunnel experiment—feedback with 2 actuators and 5 sensors [TU Braunschweig].
19. Drag reduction of an Ahmed body in a wind tunnel—feedback with 4 actuators (unison) and 12 pressures sensors) (Li et al., 2017, 2018).
20. Symmetrization of a bi-modal Ahmed body wake control in a wind tunnel—feedback with 4 actuators and 12 pressures sensors) (Li et al., 2016).
21. Drag reduction and side wind stabilization of a yawed truck model—feedback with 5 actuators and 18 sensors under transients [TU Braunschweig].
22. Jet mixing optimization with a wind-tunnel experiment—multi-frequency open-loop with 1 actuator (Wu et al., 2018).
23. Ditto but with 6 actuators (Zhou et al., 2020).
24. Vortex-induced vibration of a circular cylinder with 2 jets on the lee side working in unison and 1 sensor signal feeding back the transverse displacement (Ren et al., 2019).

All examples are based on very similar meta parameters of GPC (Duriez et al., 2017). Typically, GPC is based on linear/tree-based genetic programming with a similar structure and requires 10 generations with 100 individuals. The genetic operations include elitism (1 individual), replication (10% probability), mutation (20% probability) and crossover (70% probability). No meta parameter tuning seems necessary for typical applications. The GPC can be taken 'off the shelves" and expected to work with the standard implementation.

This book is organized as follows. Chapter 2 introduces the flow control problem as an optimization problem and describes the linear genetic programming methodology for function optimization. Chapter 3 constitutes the user guide. The chapter contains the main commands to get started with the code as well as a description of the content of the code and a list of useful commands. The commands are exemplified on the stabilization of a Landau oscillator. The `xMLC` code is then demonstrated on the net drag reduction of the fluidic pinball in chapter 4. Three types of controllers are optimized: First, multi-frequency forcing; Second, feedback control; Thirdly, an hybrid control combining multi-frequency forcing and feedback control. Finally, chapter 5 concludes this book and opens on the future of machine learning control.

Chapter 2

Machine learning control

This chapter describes the machine learning control framework and the methodology for control law optimization. First, the flow control problem is described as a regression problem (Sec. 2.1), then the linear genetic programming control (LGPC) algorithm is detailed (Sec. 2.2), finally, the main parameters for the optimization are summarized.

2.1 Flow control as a regression problem

A flow control problem consists on deriving the optimal control law $\boldsymbol{K}^*$ that fulfills a given objective, such as drag minimization, lift increase, noise reduction, mixing enhancement, etc. A control law is a mapping between the inputs ($\boldsymbol{b}$) and the outputs ($\boldsymbol{s}$) of the system to control:

$$\boldsymbol{b} = \boldsymbol{K}(\boldsymbol{h}, \boldsymbol{s}). \tag{2.1}$$

In equation (2.1), the different quantities are:

- $\boldsymbol{b}$ is the actuation command, i.e., the signal sent to the actuators;
- $\boldsymbol{s}$ is the sensor vector that comprises the measurements provided by the sensors;
- $\boldsymbol{h}$ is a vector comprising time-dependent functions such as periodic functions (see Sec. 4.3).

A key enabler for employing machine learning techniques to derive the optimal control law $\boldsymbol{K}^*$ is to reformulate the problem as a regression problem. In this framework, the control objective is translated in a *cost function* or the *loss function*, J, to be minimized. The typical control problem can be expressed as a multi-objective minimization problem. Let J be the cost function to minimize according to the control law $\boldsymbol{K}$. J is then the sum of different control objectives that quantifies the control efficiency, it reads:

$$J = J_a + \sum_k \gamma_k J_k \tag{2.2}$$

where J_a is the quantity to minimize, e.g. drag power, distance to given state, etc. The J_k are the secondary objectives to be interpreted as optimization constraints. Their contribution is weighted by the coefficients γ_k. The most common control problem comprises only two terms, J_a and an actuation penalization term J_b such as the cost function is reduced to:

$$J = J_a + \gamma J_b \tag{2.3}$$

The control problem can now be reformulated as an regression problem:

$$\boldsymbol{K}^* = \underset{\boldsymbol{K} \in \mathcal{K}}{\arg\min}\, J(\boldsymbol{K}) \tag{2.4}$$

where $\mathcal{K} : A \mapsto B$ is the space of all possible control laws, also referred as *control law space*. Here A is the input space, e.g., the space of sensor signals and B the actuation range. The regression problem (2.4) is a hard, non-convex problem, possibly including many minima. In this book, we introduce the software `xMLC` based on a linear genetic programming to learn a control law $\boldsymbol{K}$ that minimize the cost function J. Figure 2.1 gives an overview of the system to control and the role of `xMLC` in the learning loop.

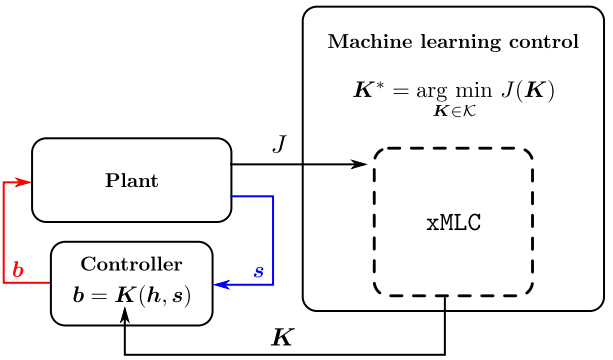


Figure 2.1: Control problem overview. The plant is controlled by actuators and monitored by sensors. The control can be carried out in an open- or closed-loop manner. In closed-loop control, the controller sends the actuation commands $\boldsymbol{b}$ to the plant based on sensor signals $\boldsymbol{s}$. The software `xMLC` builds a control law $\boldsymbol{K}$ that minimizes the cost function J. The control law is optimized through 'trial and error' based on its performance.

2.2 Linear genetic programming control

Machine learning control (MLC) based on genetic programming control (Dracopoulos, 1997, GPC) is an evolutionary algorithm for function optimization. MLC relies on biological-inspired mechanisms to build candidates solutions for the regression problem (2.4) in an iterative and stochastic manner.

The main idea of evolutionary algorithms is based on the evolution of a set of candidate solutions throughout generations thanks to selected recombinations. Following the biological terminology, a candidate solution is also called an individual and a set of individuals, a population. The optimization process relies mainly on three evolutionary principles:

The survival of the fittest: it is the selection of the most fitting individual, or the most efficient individual according to the environment, to form the next generation. This mechanism allows that the features of the 'best' individuals pass to the next generation to build, eventually, better individuals;

Crossover: it is one of the two forces of evolution that brings diversity to the population and gives opportunity to improve individuals; crossover is able to exploit the strengths of individuals by recombining two or more individuals and generating one or more offspring build from their 'parents'.

Mutation: it is the second force of evolution; it is the force that brings novelty to the population; new and more better features are likely to appear thanks to mutation.

It is worth noting that both crossover and mutation are stochastic mechanisms. Indeed, the recombination and the mutation of given individuals are random processes that, in general, give always different results. When solving a regression problem with an evolutionary algorithm, the environment corresponds to the cost function J; It assess the performance/quality of an individual. Mutation and crossover are referred as *genetic operators*.

In the following, we describe the genetic programming control algorithm in the control framework. First, we present the internal representation of the control laws and how we operate on them to generate new control laws.

2.2.1 Control law representation

To be able to combine and mutate the control laws throughout the generations, an internal representation of a mathematical function is needed. The present software `xMLC` is based on linear genetic programming (LGP) and adopts a matrix representation for the control laws (Brameier and Banzhaf, 2006).

In LGP, the individuals are considered as little computer programs, using a finite number N_{Instr} of instructions, a given register of variables and a set of constants. The instructions employ basic operations ($+$, $-$, $\times$, $\div$, cos, sin, tanh, etc.) using inputs (h_i time-dependent functions and s_i sensor signals) and yielding the control commands as outputs. A matrix representation conveniently comprises the operations of each individual. Every row describes one instruction. The first two columns define the register indices of the arguments, the third column the index of the operation and the fourth column

the output register. Before execution, all registers are zeroed. Then, the last registers are initialized with the input arguments, while the output is read from the first registers after the execution of all instructions. This leads to a $N_{\text{Instr}} \times 4$ matrix representing the control law $\boldsymbol{K}$. The name 'linear' refers to the sequential execution of the instructions. If the operation to execute only requires one operand, only the first column is considered and the second one is ignored. Each column of the matrix has its own range of values following what it codes. For single input control, i.e. when there is only one controller, the control law is read in the first register. For N_{b} controllers, the control laws are read in the first N_{b} registers. Finally, to avoid definition problems, the operators such as division and logarithm are protected to be defined on $\mathbb{R}$ the space of all the real numbers, see Duriez et al. (2017).

The registers play the role of memory slots. We distinguish two types of registers:

Variable registers: they are registers that can be overwritten while executing an instruction. They help to store intermediate results.

Constant registers: they are registers that are protected during the reading of the matrix. They are used to store random constants or data of the problem.

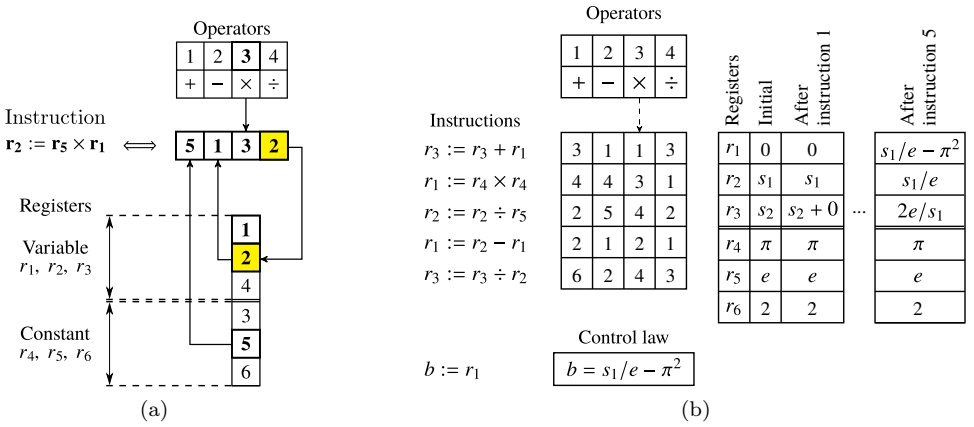


Figure 2.2: (a) Matrix representation of one single operation. The representation is inspired of reverse polish notation. The vertical column are the indices of the registers. The result of the instruction is stored in the variable register r_2, overwriting its previous value. (b) Transcription of a matrix of instructions into a control law (expression framed in the bottom). The matrix (middle) has five instructions. The instruction are displayed on the left and the evolution of the registers after the execution of the first and fifth instruction on the right. The library of operators ($+$, $-$, $\times$, $\div$) and there corresponding index are displayed on the top. The first three registers (r_1, r_2, r_3) are the variable register and the last three (r_4, r_5, r_6) are the constant register. The control law is derived from the expression stored in the first register. In case of multiple input problems the others control laws would be derived from the following registers.

Figure 2.2a, illustrates how a single instruction is represented in matrix form.

It is worth noting that for a given mathematical expression, there is more than one matrix representation. Indeed, as stated before, there are instructions in the matrix that have no impact in the output registers. Also, the matrix representation takes into account the order of the operations even for operators that are commutative: the control laws $b = s_1 + s_2$ and $b = s_2 + s_1$ will be have different representation while being the same mathematical expression. As consequence, several instances of the same individual can be present in the population. In order to accelerate the learning, such individuals are removed. To detect them, the control laws are all evaluated on the same random samplings. If the result of the evaluation of one individual correspond to a previous evaluation, then the individual is replaced.

Figure 2.2b depicts how a matrix of instructions is read to build a control law. We notice that in the instruction matrix, not all instruction lines are useful. Indeed, if an instruction line does not affect one of the output registers then it is, in reality, useless. However, in the process of recombination or mutation, these instructions lines can be 'activated', changing the final control law. Following Brameier and Banzhaf (2006), these 'useless' instruction lines, also called *introns*, play a major in the process of building relevant structures.

With enough instructions and operators, any function can be represented in matrix form. LGP can, for example, reproduce the Taylor expansion of any function until an arbitrary order by deriving the

coefficients of the power series. Also, using the matrix representation, we do not constrain, a priori, the structure of the control laws. Of course, the solutions built strongly depend on the library of operators and control inputs given to the algorithm. Indeed, the richness of these libraries defines the complexity of the search space for the regression problem. The choice of the function libraries is studied in Sec. A.3 on a dynamical system whereas different sets of control inputs are tested for the control of the fluidic pinball in Sec. 4.1.

Before giving the genetic algorithm in its final form, we first describe its first, the Monte Carlo step, as it is an optimization algorithm on its own.

2.2.2 Monte Carlo sampling

A starting point for genetic programming is the random generation of the first set of individuals. This operation can be seen as a Monte Carlo sampling process. In the LGPC framework, to define a control law is to chose a library for the operators ($+$, $-$, cos, etc.), a library for the inputs (a_1, a_2, etc.), the maximum number of instruction $N_{\text{Instr,max}}$, or the number of rows in the matrices, the number of variable registers N_{Var} and the number of constant registers N_{Cst}. From these parameters, we can generate random matrices that are then read sequentially to form control laws. The number of instructions for each matrix is randomly drawn from an uniform distribution between 1 and $N_{\text{Instr,max}}$. In theory, a Monte Carlo process is enough to solve equation (2.4) but a very large number of individuals might be needed to reach the global optimum of the problem, especially for search spaces of infinite dimension. For pragmatic reasons and also to emulate limited experiment time, we fixed the total number of individuals tested N_i. N_i can also be seen as the total number of cost function callings and also the total number of experiments to run. Figure 2.3 illustrates the Monte Carlo sampling process. N_i individuals are generated randomly,

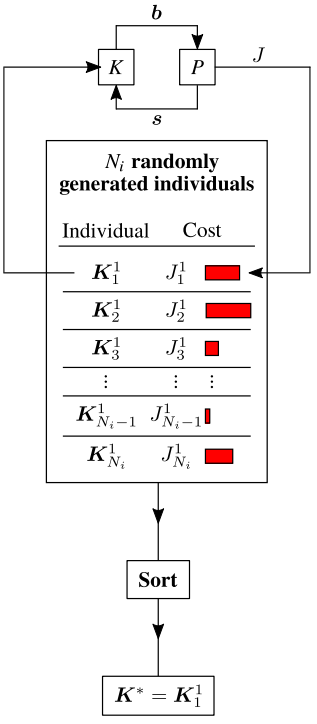


Figure 2.3: Monte Carlo sampling. A set of randomly generated control laws is evaluated and sorted following their performances. The red bar on the right of each individual represents their performance. The smaller the bar, the better the individual performs. The superscript 1 on the individuals $\boldsymbol{K}_i$ means that they belong to the first generation. The top part of the figure, represents the controller K, receiving the sensors signals $\boldsymbol{s}$ as input and giving back the actuation command $\boldsymbol{b}$ to the plant P.

they are all tested and sorted following their cost. The final result of the algorithm is the individual with the lowest cost J, thus the most performing following the cost function criterion. Figure 2.3 depicts the Monte Carlo process for controlling a plant P (framed in the figure).

In the next section, we describe how to create the next generations of individuals from a set of individuals generated thanks to a Monte Carlo process.

2.2.3 The evolution process—Selection and genetic operators

In the following sections, we describe how the natural selection is emulated and the implementation of the genetic operators. In this section, we detail the steps carried out to create a new population based on a previous one.

Selection

To create the next generation of individuals, we need, first, to select the most performing individuals to be combined and mutated. The operation of selection is carried out thanks to a tournament selection. The idea of a tournament selection is to select N_{Tour} individuals among the N_i individuals in the population. Among the N_{Tour} individuals selected, the best one is selected with a probability of P_{Tour}. If the best one is not chosen, the second best is chosen with the same probability P_{Tour} and so on for all the selected individuals. At the end, if no other individual is chosen, the least performing among the N_{Tour} is selected. The choice of N_{Tour} and P_{Tour} influence the extent to which well-performing individuals are preferred over least-performing ones. This feature is called *selection pressure* and is developed in detail in Wahde (2008).

Crossover for exploitation

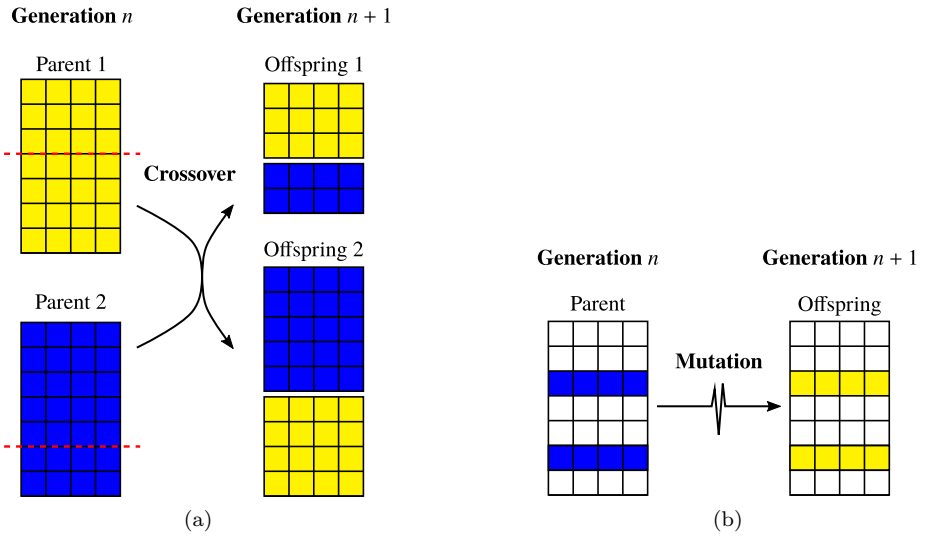


Figure 2.4: (a) Exploitation of the genetic material by recombining two individuals. The parts of the matrices are switched to build new matrices. (b) Exploration of the search space by creating new instructions and thus new structures.

Crossover is the operation of recombination of individuals. It has the potential to extract and combine relevant structures in the individuals. That is why, we refer this genetic operator as the *exploitation operator*. To combine the individuals, two individuals are selected in the population, and their matrices are split in two and the parts are swapped to generate two new individuals, also referred as *offsprings*. Figure 2.4a illustrates the crossover operation between two individuals.

It is worth noting that, the crossover operation is defined such as the length of the matrices may increase or decrease. To avoid that the size of the matrices explodes, we set a upper limit to the number of rows in the matrix. In practice, this limit is the same $N_{\text{Instr,max}}$. If this limit is exceeded, then the operation is restarted until offspring with lesser instructions are built.

Mutation for exploration

Mutation is the operator that generates new sequences in the matrices. The role of this operator is to find new structures, unknown to the population, to improve the solutions. For the mutation of one individual, each row of the corresponding matrix representation has a probability of P_{Mut} to be completely changed. The parameter P_{Mut} is chosen such as at least one row is changed in the matrix. The change of one line can either have no consequences in the final output, if the instruction stays an intron, or it can also completely change the final output. To improve our exploration potential, we choose to restart the

mutation operation when the mutated individual is identical to the original one. Figure 2.4b, depicts the process of mutation for an individual.

Of course, there are several ways to define the crossover and mutation operators but we choose to realize the simplest implementation.

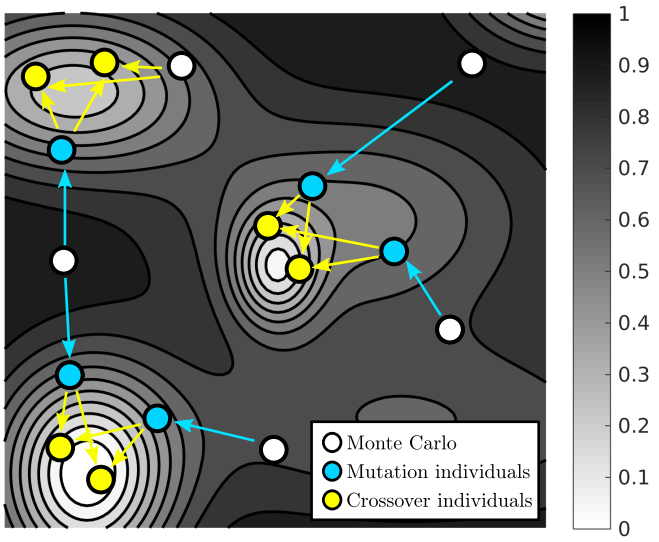


Figure 2.5: Principle sketch of the exploitation/exploration potential of crossover and mutation. The background is a 2D representation of a control landscape. White regions denoting good performances and dark regions poor performances. Three minima are depicted. After an a priori exploration of the control landscape with Monte Carlo, crossover and mutation improve the evaluated individuals. Crossover individuals have the potential to explore the neighborhood of a minimum whereas mutation discovers new minima by combining good individuals.

From an optimization point of view, crossover is the operator that improves existing solution. Its role is to 'explore' the neighborhood of a minimum, while mutation is the one that explores the control landscape to discover new minima. The learning principles are illustrated in figure 2.5.

Replication and elitism for memory

In addition to crossover and mutation, we also consider two other operators: replication and elitism. With replication an identical copy of one individual is copied to the next generation, assuring memory of good individuals and allowing future recombination. This elitism operation assures that the bests individual are always in the latest generation so that 'the winner does not get lost' throughout the generations. In this case, the individuals are simply copied to the next generation. The number of individuals selected by elitism is defined by the elitism parameter N_{E}, usually it is set to $N_{\mathrm{E}} = 1$.

Figure 2.6, illustrates the complete LGPC algorithm. The first generation of individuals is generated thanks to a random sampling of the individuals (Monte Carlo method). Then, from a generation n, the individuals are all evaluated and sorted following their performances. The best individuals are then selected, thanks to a tournament method, to be modified and recombined with crossover and mutation. Replication and elitism assures a memory of the good individuals. The choice of crossover, mutation or replication to populate the next generation is controlled by the probabilities P_{Cros}, the crossover probability, P_{Mut}, the mutation probability and P_{Rep} the replication probability. They are chosen such as $P_{\mathrm{Cros}} + P_{\mathrm{Mut}} + P_{\mathrm{Rep}} = 1$. The balance between crossover, mutation and replication is thoroughly analyzed in App. A.2.

There are several variations of the genetic programming algorithm, where the genetic operators are not separated but applied one after the other and where the offspring replaces the parent individual in the population only if it performs better. In this study, we choose to follow the classical evolutionary algorithm described in Brameier and Banzhaf (2006) and also employed by Duriez et al. (2017).

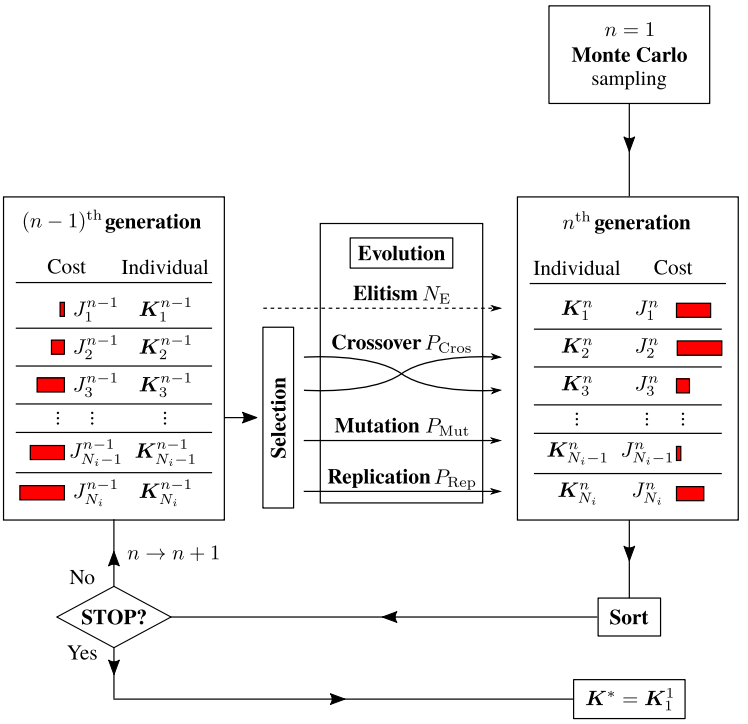


Figure 2.6: Linear genetic programming algorithm employed in `xMLC`.

In the next section, we illustrate the learning process of LGPC by stabilizing a Landau oscillator with `xMLC`.

Chapter 3

User guide

This chapter contains all the elements to download, install and run the xMLC software on a toy system. This chapter is recommended for a quick start with xMLC. First, the mean features of the software are described (Sec. 3.1), then the requirements and installation are detailed (Sec. 3.2). Thereafter, the main commands for a learning process are exemplified on the control of a toy system (Sec. 3.3. Finally, an overview of the MLC MATLAB class is given and how to change the main parameters (Sec. 3.4). The user who wants to employ xMLC to her/his own control problem is invited to follow the guide in App. B to write her/his own parameter file. However, it is suggested to run the commands in this section first to get familiar with code.

3.1 xMLC features

The xMLC software is an implementation of the linear genetic programming algorithm for control optimization. The software allows in particular for:

Open-loop control: Control laws only depending on time-dependent functions can be optimized to derive, for example, a multi-frequency controller combining different periodic functions.

Closed-loop control: Of course, closed-loop control laws can be learned by including sensor signals in the input library.

Hybrid control: An hybrid optimization is also possible by combining sensor information and time-dependent functions.

From SISO to MIMO control: xMLC is able to optimize control laws for single-input single-output (SISO) systems as well as multiple-input multiple-output (MIMO) systems. There is no limit on the number actuators and sensors one can use.

In addition of the optimization process, xMLC includes a variety of post-processing tools to analyze the learning process and visualize the distribution of evaluated individuals, such as learning curve, extraction of the best individual and Pareto front. Some of these features are displayed in the next section (Sec. 3.3). Moreover, xMLC includes a set of scripts and commands for a interfacing the optimization code with numerical simulations on computer clusters and experiments. Finally, the code is not restricted to control problems as it can be adapted to solve any function regression problem.

3.2 Download and installation

In this section we present the necessary steps to download and install the xMLC software.

Requirements

xMLC is available for both MATLAB and Octave. It has been coded on MATLAB version 9.5.0.944444 (R2018b) and Octave version 4.2.2. any further version should be compatible with the software. No particular MATLAB or Octave package is needed for the proper functioning of the software.

Installation

The `xMLC` software can be download from the following links

`https://doi.org/10.24355/dbbs.084-202208220937-0` or `https://github.com/gycm134/xMLC`

under the MIT License (MIT).

Once downloaded, decompress the tar file and copy the `MLC/` folder where it is needed. Installation is then complete. For further information on the content of the `MLC/` folder please look at Sec. 3.4 and the README.md file.

3.3 Quick start

In this section, we present the main commands to optimize control laws. The process is illustrated by stabilizing the damped Landau oscillator, a dynamical system with a stable limit cycle. The commands are executed on MATLAB but they can be also ran on Octave. Only the methods containing MATLAB in their name needs to be replaced by their Octave counterparts.

3.3.1 The damped Landau oscillator

The controlled dynamical system

The damped Landau oscillator is a system of two coupled ordinary differential equations with a nonlinear damping of the growth rate. Despite its simplicity, it describes a fundamental oscillatory process at the heart of physical mechanisms such as the von Kármán vortex shedding behind a cylinder (Luchtenburg et al., 2009). To control the oscillator a forcing term b is introduced in the second equation. The systems reads:

$$\left\{\begin{array}{lcl} \dot{a}_1 &=& \sigma a_1 - a_2 \\ \dot{a}_2 &=& \sigma a_2 + a_1 + b(a_1, a_2) \\ \sigma &=& (1 - a_1^2 - a_2^2) \end{array}\right. \tag{3.1}$$

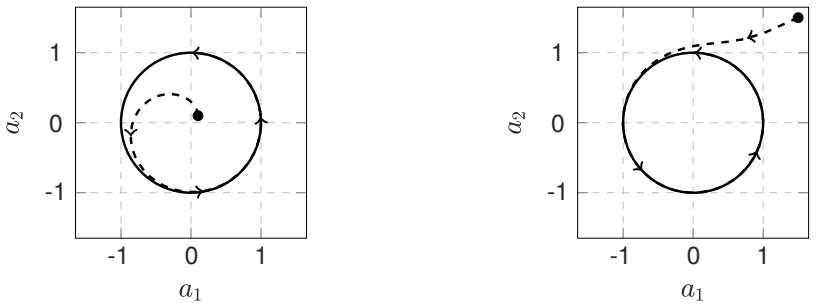


Figure 3.1: Phase portrait of the oscillator with no actuation ($b = 0$) with $(a_1, a_2)_{t=0} = (0.1, 0.1)$ as initial condition for (a) and $(a_1, a_2)_{t=0} = (1.5, 1.5)$ as initial condition for (b). In both cases the system converges towards the limit cycle of radius 1.

For $b = 0$, we have an oscillator of growth rate σ, angular frequency 1, period $T = 2\pi$ and fixed point $(0, 0)$. For an initial condition close to the fixed point, the quadratic terms in σ are negligible, leading to an exponential growth. When the system is far from the fixed point, the growth is damped due to the quadratic terms, stabilizing the oscillator to the limit cycle of radius $\sqrt{1} = 1$. The same reasoning for an initial condition outside the circle of radius 1 shows that the limit cycle is globally stable. The uncontrolled dynamics are depicted in figure 3.1a and 3.1b. The control on the second equation has the effect of pushing the system upwards or downwards following the sign of b: Upwards if $b > 0$ and downwards if $b < 0$.

Objective and cost function

The control objective is to bring the system to the fixed point $(a_1, a_2) = (0, 0)$ from the limit cycle. Two terms are considered for the cost function: J_a, the averaged distance to the fixed point and J_b, an

actuation penalization term.

$$\begin{aligned} J &= J_a + \gamma J_b \\ J_a &= \frac{1}{T_{max}} \int_0^{T_{max}} (a_1^2 + a_2^2) d\tau \\ J_b &= \frac{1}{T_{max}} \int_0^{T_{max}} b^2 d\tau \end{aligned} \tag{3.2}$$

The penalization parameter γ is taken equal to 0.01, such as the optimization process focuses first on the primary objective, i.e., stabilizing the oscillator. Both are integrals quantities over 10 periods which corresponds to $T_{\max} = 20\pi$ as we are interested not only in the final solution but also on the trajectory. In order to assure a general solution, we consider four initial conditions on the limit cycle: $(1, 0)$, $(0, 1)$, $(-1, 0)$, $(0, -1)$. The cost function J is then a mean value between these four initial conditions. For the uncontrolled dynamics, we have $a_1^2 + a_2^2 = 1$ and $b^2 = 0$, therefore $J_a = 20\pi \approx 62.83$ and $J_b = 0$. The unforced cost is then $J_0 = 20\pi$.

In the `xMLC` software, the damped Landau oscillator system and its resolution is implemented in the `Plant/LandauOscillator/LandauOscillator_problem.m` file. See Sec. 3.4 for more information and the content of the code.

3.3.2 First run

We now present the main commands to quickly use the `xMLC` software.

Initialization

To use the MLC software launch a MATLAB session on the `MLC/` folder. We then follow the steps of the `CheatSheet.m` file, providing the corresponding outputs and figures. It is advised to not execute the whole `CheatSheet.m` script but rather execute it section by section. The first step is to launch the `Initialization.m` script to load all the necessary paths. In the following insert, we include the command and the output.

```
>> Initialization;
====================== xMLC v0.10 ====================
 Welcome to the xMLC software to solve function
 regression problems.
 In case of error please contact the author :
  Guy Y. Cornejo Maceda Website
 The MIT License (MIT)

 Start by creating a MLC object with : mlc=MLC;
=======================================================
```

The command output gives a short description of the `xMLC` code and invites us to create a MLC object. The MLC object is in fact a structure array containing the parameters, the database and information on the current generation. Let's create such MLC object:

```
>> mlc=MLC;
====================== xMLC v0.10 ====================
Name of the run : TestRun
Problem to solve : LandauOscillator
Problem type : MATLAB
   Number of actuators       : 1
   Number of control inputs  : 2

Parameters :
   Population size : 10
   Elitism : 1
   Operator probabilities :
      Crossover : 0.600
      Mutation : 0.300
      Replication : 0.100

To generate a population : mlc.generate_population;
To run N generations : mlc.go(N);
=======================================================
```

The command output gives information on the problem and parameters in the MLC object. The 'Name of the run' is the name of the MLC object. When an optimization is initiated, all its associated files are stored in a folder with the name of the run. For this example, the folder `save_runs/` will be automatically created along with the `save_runs/TestRun/` folder.

The problem set by default is the stabilization of the Landau oscillator described in Sec. 3.3.1; it is implemented in the `Plant/LandauOscillator/LandauOscillator_problem.m` file. For this example, the system of equations (3.1) including the control is solved directly by MATLAB. The `xMLC` software can also be interfaced with other solvers or experiments, see App. C for more information.

The output also displays the main parameters. First, the number of actuators and control inputs are displayed In this case, there is only one actuator, the forcing term b in the second equation and its inputs are a_1 and a_2. Then, the main optimization parameters, population size, elitism parameter and genetic operator probabilities. All these parameters are defined in the `MLC_tools/default_parameters.m` file. Here are some examples on how to modify these parameters:

```
>> mlc.parameters.Name = 'AQuickTest';
>> mlc.parameters.PopulationSize=100;
>> mlc.parameters.Elitism = 1;
>> mlc.parameters.CrossoverProb = 0.6;
>> mlc.parameters.MutationProb = 0.3;
>> mlc.parameters.ReplicationProb = 0.1;
```

Note that these parameters are modified only for the current MLC object. The user can also modify the initial parameters by creating her/his own parameter file, see App. B.

Control law optimization

Once the MLC parameters are appropriately set, the optimization process can be begin. There are two ways to advance the optimization process: either running the commands one after the other or using the `go` method. Let's look at the first method. For this, we follow the instructions displayed on the command prompt and generate a population with the appropriate command:

```
>> % Create the first generation
>> mlc.generate_population;
Generating new population
    Population size = 100
    Generating new individual 1/10 Done.
    Generating new individual 2/10 Done.
    Generating new individual 3/10 Done.
    ...
    Generating new individual 98/100 Done.
    Generating new individual 99/100 Done.
    Generating new individual 100/100 Done.
End of generation : population generated in 3.4128 seconds.
No pre-evaluated individuals to be removed.

To evaluate the population: mlc.evaluate_population;
```

The output displays the creation process of the individuals. Once an individual or a control law is created it is evaluated on a random set of inputs, if this evaluation returns `INF` or `NAN`, then the control law is discarded an a new is generated. In this example, all the individuals have been properly evaluated thus the message: 'No pre-evaluated individuals to be removed'. Now, the population can be evaluated with the corresponding command:

```
>> % Evaluate the first generation
>> mlc.generate_population;
Evaluation of generation 1 :
    Evaluation of individual 1/100  J= 62.831878
    Evaluation of individual 2/100  J= 75.394162
    Evaluation of individual 3/100  J= 59.252250
    ...
    Evaluation of individual 98/100  J= 90.154683
    Evaluation of individual 99/100  J= 46.342791
    Evaluation of individual 100/100  J= 78.925153
No bad individuals to be removed.
```

```
To create the next generation : mlc.evolve_population;

To create the next generation : mlc.evolve_population;
```

The cost of each individual is displayed next to its evaluation message. The message 'No bad individuals to be removed' means that all the individuals have been properly evaluated. Otherwise, the code attributes to them a high cost ($\approx 10^{36}$) so that the selection process automatically eliminates them. The default parameters are such as the code replaces those 'bad' individuals with new ones. The replacement of bad individuals is repeated until the population does not include any of them. This option is harmless when the evaluation of the individuals is 'quick' but it is advised to deactivate it for time-consuming evaluations or experiments. This option can be deactivated with the command before launching the optimization process:

```
>> % Deactivate the recursive removal of individuals whose evaluation failed.
>> mlc.parameters.RemoveBadIndividuals = 0;
```

Let's generate the next generation and evaluate it with the corresponding commands:

```
>> % Generate the next generation (Gen 2)
>> mlc.evolve_population;
Generation 2
    Generating new individual 1/10 Elitism.
    Generating new individual 2/10 Crossover.
    Generating new individual 3/10 Crossover.
    ...
    Generating new individual 98/100 Replication.
    Generating new individual 99/100 Mutation.
    Generating new individual 100/100 Mutation.
No pre-evaluated individuals to be removed.

To evaluate the population: mlc.evaluate_population;
>> % Evaluation of the new generation (Gen 2)
>> mlc.evaluate_population;
Evaluation of generation 2 :
    Evaluation of individual 1/100 already done (J= 32.039854)
    Evaluation of individual 2/100  J= 60.631446
    Evaluation of individual 3/100  J= 64.893979
    ...
```

Note the presence of the genetic operator that generated the new individual. During the evaluation of the second generation, the individuals that have already been evaluated are not re-evaluated. One can force the re-evaluation with the parameter `MultipleEvaluations`, see App. B. We could continue the optimization process by alternating the evolution and evaluation commands but there is a simpler way. One can directly generate and evaluate the next generation with the `go` method and also directly indicate the final generation.

```
>> % Generate and evaluate the next generation (Gen 3)
>> mlc.go;
Generation 3
    Generating new individual 1/100 Elitism.
    Generating new individual 2/100 Replication.
    Generating new individual 3/100 Crossover.
    ...
 Evaluation of generation 3 :
    Evaluation of individual 1/100 already done (J= 32.004032)
    Evaluation of individual 2/100 already done (J= 51.542092)
    Evaluation of individual 3/100  J= 53.512292
    ...
1 Generation(s) computed in 25.5563 seconds
 Cost of the best individual = 3.1794
 Its expression =
    b = (tanh((s(1) - s(2))) - s(2))
```

When the `go` is employed it also gives back information on the best individual evaluated so far. In this case, the best control law is `tanh(((s(1) - s(2)) - s(2)))` which after simplification corresponds to: $\tanh(a_1 - 2a_2)$. To continue the optimization until generation 10, run the command:

```
>> % Generate and evaluate the next generation (Gen 3)
>> mlc.go(10);
    ...
Evaluation of individual 98/100 already done (J= 3.730651)
    Evaluation of individual 99/100  J= 5.254885
    Evaluation of individual 100/100  J= 22.423008
No bad individuals to be removed.

To create the next generation : mlc.evolve_population;
7 Generation(s) computed in 228.2561 seconds
 Cost of the best individual = 1.3074
 Its expression =
    b = my_div(tanh(my_div(tanh(my_div(tanh((s(1) - s(2))),0.59655)),0.59655))
       ↪  ,0.59655)
```

Once the process is over, the MLC object can be saved to continue the optimization or post-processing later.

```
>> % Save the run
>> mlc.save_matlab;
```

For this example, the MLC object is saved in `save_runs/AQuickTest/MLC_Matlab.mat`. To continue the optimization later or in a new MATLAB session, first create a MLC object then use the `load_matlab` method with the name of the run.

```
>> % To load an existing run on a new MATLAB session.
>> % First create a MLC object.
>> mlc=MLC;
>> % Then load the run with its name
>> mlc.load_matlab('AQuickTest');
```

For Octave, employ the corresponding methods `save_octave` and `load_octave`. `xMLC` also allows intermediate savings of the same MLC object, for more information see Sec. 3.4.

Post-processing and analysis

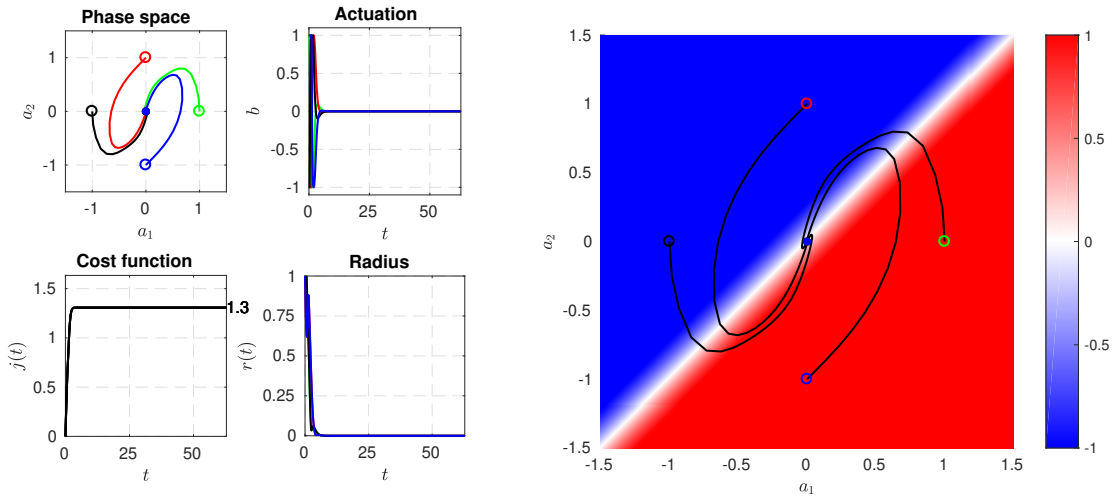
Once the optimization process is done, the best individual can be accessed with the following method :

```
>> mlc.best_individual;

Best individual after 10/10 generations (ID:3019):
   Its cost : 1.307
   Control law : my_div(tanh(my_div(tanh(my_div(tanh((s(1) - s(2))),0.59655))
      ↪  ,0.59655)),0.59655)
   Number of instructions (effective): 15 (7)
```

The output mentions that the best individual has been found in the 10^th generation over the 10 generations evaluated. The ID number represents its index in the database. The number of instructions corresponds to the number of lines in its matrix representation. The number in parenthesis indicate that among the 15 lines of the matrix only 7 contribute to the final expression of the control law. The `give` method also gives a figure displaying the performance of the best individual, see figure 3.2. The code for the figure is included in the `Plant/LandauOscillator/LandauOscillator_problem.m` file.

For more information on the individual, run the command:

```
>> mlc.give(3019);

3019-th Individual of the database:
   cost (mean over 1 evaluations): 1.307428
      1-st evaluation cost : 1.307428
   occurrences: 1
   control law:
      b1 = my_div(tanh(my_div(tanh(my_div(tanh((s(1) - s(2))),0.59655))
         ↪  ,0.59655)),0.59655)
```



(a) Phase space, actuation command, instantaneous cost function and radius for the controlled case.

(b) Visualization of the control law in the phase space.

Figure 3.2: Figures characterizing the control law optimized after 10 generations for the stabilization of the Landau oscillator. The figures have been generated with the `best_individual` method.

```
   ref:0

>> % Check the informations of the individual in the database.
>> % The matrix representation.
>> mlc.table.individuals(3019).chromosome;
>> % Cost of the individual: J, Ja, Jb.
>> mlc.table.individuals(3019).cost;
>> % Control law.
>> mlc.table.individuals(3019).control law;
```

Of course these commands can be executed for any ID in the database. Note that the matrix is referred as chromosome in `xMLC` to follow the evolutionary terminology. Other features can be extracted such as the learning process (see figure 3.3a), the Pareto diagram (see figure 3.3b), the cost distribution (see figure 3.3c) and the spectrogram (see figure 3.3d) thanks to the following commands:

```
>> % Plot of the learning process.
>> mlc.learning_process;
>> % Plot of the Pareto diagram and front for the first components of the cost
   ↪ function.
>> mlc.Pareto_diagram;
>> % Plot of the cost distribution sorted by J.
>> mlc.cost_distribution;
>> % Plot of the distribution of costs in each generation.
>> mlc.spectrogram;
```

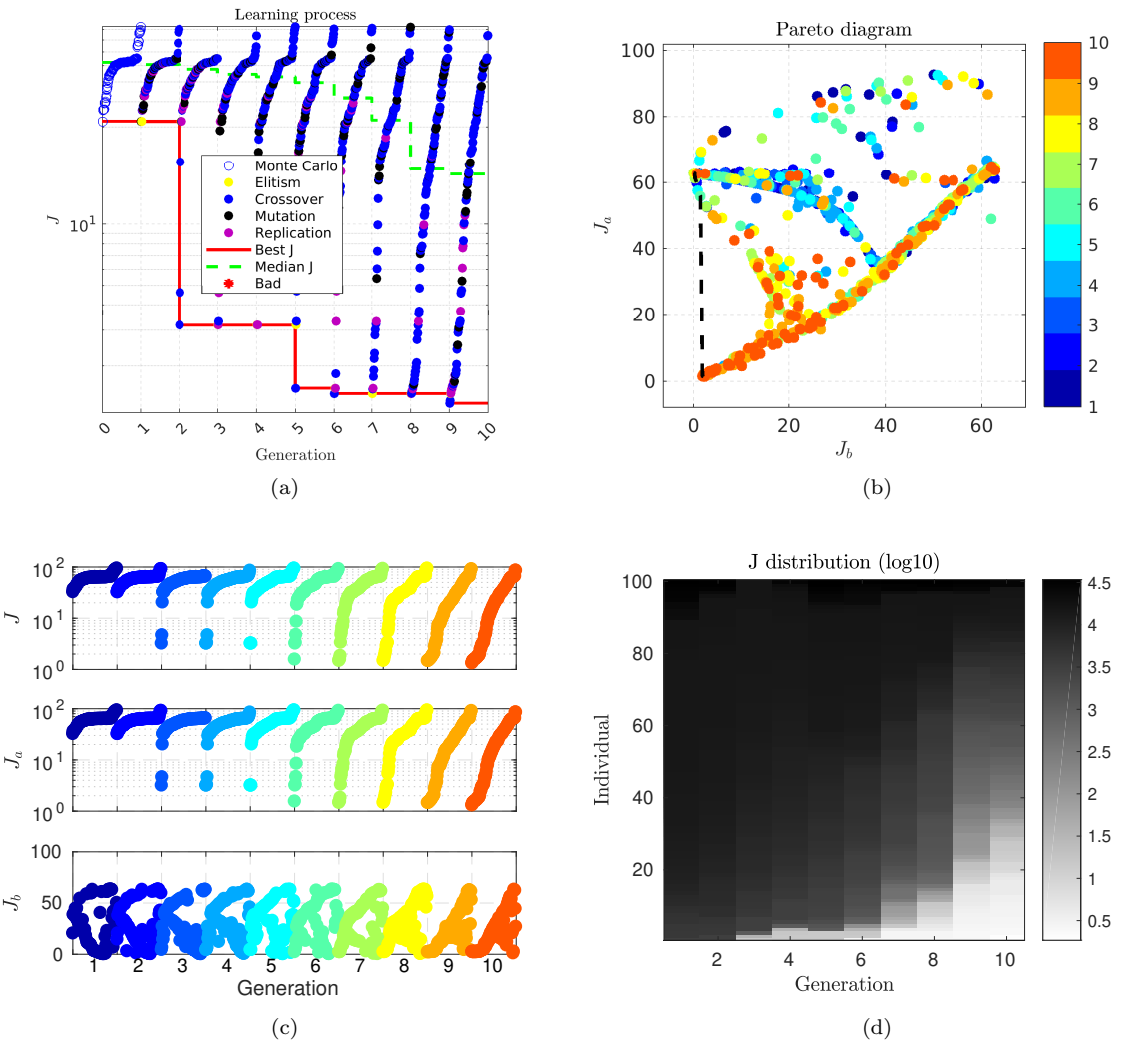


Figure 3.3: Figures characterizing the optimization process and displaying the distribution of the control laws evaluated. Figure (b) and (c) share the same color code for the generations. The figures have been generated with specific methods: (a) `learning_process`, (b) `Pareto_diagram`, (c) `cost_distribution` and (d) `spectrogram`.

3.4 Code description

In this section, we give a brief description of the `xMLC`. First, we detail the content of the code then we give an overview of the MLC class and its properties. Finally, we provide a list of useful commands for the analysis and extraction of information from the MLC object.

3.4.1 Content

Figure 3.4 displays the content of the main folder `xMLC/`. In the following, we give a description of each element of the folder.

- The folder `@MLC/` containing the file `MLC.m` that defines the MLC class with its properties and methods;
- The folders, `@MLCind/`, `@MLCpop/`, `@MLCtable/`, defining three other classes that are employed in the MLC class: MLCind, MLCpop and MLCtable define an individual, a population and a database respectively;
- The folder `MLCtools/` containing additional functions employed during the optimization;

```
MLC/ ..... MLC class
├── MLCind/ ..... Class for an individual
├── MLCpop/ ..... Class for a population
├── MLCtable/ ..... Class for the database
├── MLC_tools/ ..... Additional functions
├── ODE_Solvers/
├── Plant/ ..... Contain the different problems
├── save_runs/ ..... Contain the savings
├── CheatSheet.m
├── Initialization.m ..... First script to run
├── LICENSE
├── README.md
├── Restart.m ..... To restart the session
└── Version.txt
```

Figure 3.4: File structure for the `xMLC/` folder.

- The folder `ODE_Solvers/` containing implementations of ODE solvers in particular Runge-Kutta methods from order 1 to 5;

```
Plant/ ..... Contains the problems
├── GMFM/
│   ├── GMFM_parameters.m
│   └── GMFM_problem.m
├── LandauOscillator/
│   ├── LandauOscillator_parameters.m
│   └── LandauOscillator_problem.m
├── Tanh_fitting/
│   ├── Tanh_parameters.m
│   └── Tanh_problem.m
├── MyPlant/ ..... See B & C
│   ├── MyExternalPlant_parameters
│   ├── MyPlant_parameters.m
│   └── MyPlant_problem.m
└── T_maxevaluation.m ..... See B
```

Figure 3.5: File structure for the `Plant/` folder.

- The folder `Plant/` containing different parameter folders for each problem. Three toy problems are included, the stabilization of the Landau oscillator, the stabilization of the generalized mean-field model and the fitting of a tanh function (see figure 3.5. Each folder contains a parameter file and problem file. The parameter file contains all the `xMLC` parameters and can also include parameters for the problem. The 'problem' represents the plant, it includes the resolution of the corresponding dynamical. This file is needed only when problems are solved by MATLAB. It is not needed when the user employs an external solver or experiment. To create a personalized problem, see App. B;
- The folder `save_runs/` containing the saving files and the outputs of the code. This folder does not exist on the original code but is created automatically if needed;
- The file `CheatSheet.m` containing the main commands to start an optimization process, see Sec. 3.3;
- The file `Initialization.m` that loads all the paths necessary for the code. This script should be used at the beginning of each session;
- The file `LICENSE` contains the license of the `xMLC` code: *The MIT License (MIT).*
- The file `README.md` gives a short description of the code along with the main commands to start an optimization process. This file keeps also track of the updates/upgrades along the versions. If you encounter any compatibility problem, contact the author at Yoslan@hit.edu.cn;

- The file `Restart.m` that clears all the variables and creates a new MLC object with the default parameters;
- The file `Version.txt` containing the version of the code. The version should be communicated in case of contact with the author.

3.4.2 Description of the MLC class

Once the `Initialization.m` script is launched, a small description of the problem to be solved is printed. It contains information about the number of inputs (controllers), the number of outputs (sensors), the population size and the strategy (genetic operators probabilities). This is shown each time a new instance of a MLC class object is created. To show it again use the `show_problem` method by using the command : `mlc.show_problem;`. The `MLC` object has 6 properties:

- `individuals` ..... List of ID numbers sorted by their cost
- `costs` ..... Cost of each individual
- `chromosome_lengths` ..... Number of instructions and effective instructions
- `parents` .....ID number of the parents
- `operation` ..... Operation that generated the individual
- `generation`
- `evaluation` ..... Status of the population: evaluated or not
- `CreationOrder` ..... Order of creation of the individuals

Figure 3.6: Properties of the MLCpop class.

- `population` is an array of MLCpop. Each one of then containing information of all the individuals of the a given generation. In particular, the 'ID' corresponds to the index of the individual in the database. Note that the ID refers to a unique matrix representation. Thus, two control laws that simplify to the same expression will have different ID numbers. However, one of them may be replaced during the optimization process. Figure 3.6 gives more information on the properties of each population;
- `parameters` is a structure object containing all the parameters for a problem. It contains in particular where all the parameters are defined, be it for the problem, the control law description or the MLC parameters;

- `individuals` .....Array of MLCind objects
- `non_redundant` ..... ID list of all 'unique' individuals
- `number` .....Total number of individuals explored
- `control_points` ..... See Sec. B
- `costlist` .....List of costs for all individuals explored

Figure 3.7: Properties of the MLCtable class.

- `chromosome` .....Matrix representation of the individual
- `costs` ..... Array of cell containing the cost J and its components for all evaluations
- `control_law` ..... Array of cell containing the control law components
- `EI` ..... Information on the effective instructions
- `occurrences`
- `evaluation_time`
- `control_points` .....See Sec B
- `ref` .....ID of the reference individual if redundant individual

Figure 3.8: Properties of the MLCind class.

- `table` is the database, it contains all the individuals explored. This number may be superior to the number of individuals evaluated as some of them are removed due to failed evaluations or screening options. Figure 3.7 and 3.8 details the content of the MLCtable and MLCind classes respectively;

- `generation` display the current generation. It is set to 0 by default when the population is empty. One generation is counted only if it has been evaluated;
- `version` contains the version of the MLC object. The structure of the parameter file may be modified with the new versions thus keeping track of the MLC object version is important.

3.5 Handy commands

We list now useful commands to extract individual information or the analyze the optimization process.

```
>> % Information on the 5 best individuals evaluated so far.
>> % If an individual is evaluated several times (in case of stochastic problem
   ↪ ),
>> % the command gives the best individuals based on an estimated performance.
>> % See the EstimatePerformance parameter in App. A.
>> mlc.list_best_individuals;
>> mlc.list_best_individuals(GEN); % Same as the previous command but up to
   ↪ generation GEN.
>> mlc.list_best_individuals(GEN,NI); % Same as the previous command but gives
   ↪ only the NI best individuals.

>> % Information on individual ID:
>> mlc.give(ID);

>> % Re-evaluation of individual ID and plot the control if possible.
>> % Only works for problems evaluated with MATLAB.
>> mlc.plotindiv(ID);

>> % Intermediate save of a run name AQuickTest:
>> mlc.save_matlab('Gen2');
>> % The saving file is then stored in save_runs/AQuickTest/Gen2_Matlab.mat
>> % Load an intermediate save of a run named AQuickTest:
>> mlc.load_matlab('AQuickTest','Gen2');

>> % Information on the sensor, constant and operation distribution for
   ↪ generation GEN.
>> mlc.CL_descriptions(GEN);

>> Distribution of genetic operators through the generations.
>> mlc.genoperatorsdistrib;

>> Relationship matrix between the parents and the offsprings.
>> mlc.relationship;

>> % Print the current figure.
>> % The figure is saved in the save_runs/AQuickTest/Figures/ folder in PNG and
   ↪  EPS format.
>> mlc.printfigure('AQuickTest')
>> mlc.printfigure('AQuickTest',1) % To overwrite an existing figure.

>> % To initiate a problem different from the default one.
>> % For example, to create a MLC object wit the GMFM problem:
>> mlc=MLC('GMFM');

>> % One can also rename a run.
>> mlc.rename('AFastTest');
AQuickTest changed to AFastTest.

>> % If one's wish to duplicate a run, just copy-paste the run folder in
   ↪ save_runs/ and change it's name.
>> % The MLC object can then be loaded with the new name.
>> % The name in the parameters will be automatically updated.
```

Chapter 4

Example: Net drag power reduction of the fluidic pinball

In this chapter, we reduce the net drag power of the fluidic pinball thanks to `xMLC` software presented in Sec. 3. For this study, we explore three different search spaces: first we look for a multi-frequency forcing controller, second we look for feedback control laws and finally we investigate an hybrid search space comprising periodic functions and sensor signals. Thus, in Sec. 4.1 we present the fluidic pinball and the regression problem. In Sec. 4.2, we detail an open-loop control study of the fluidic pinball for a control reference. Lastly, we apply MLC to minimize the net drag power with multi-frequency forcing (Sec. 4.3), feedback control (Sec. 4.4) and a search space allowing both strategies (Sec. 4.5).

4.1 The fluidic pinball—A benchmark flow control problem

In this section, we describe the fluid system studied for the control optimization—the fluidic pinball. First we present the fluidic pinball configuration and the unsteady 2D Navier-Stokes solver in Sec. 4.1.1, then the unforced flow spatio-temporal dynamics in Sec. 4.1.2 and finally the control problem for the fluidic pinball in Sec. 4.1.3. This section is largely inspired from section 2 of Cornejo Maceda (2021).

4.1.1 Configuration and numerical solver

The test case is a two-dimensional uniform flow past a cluster of three cylinders of same diameter D. The center of the cylinders form an equilateral triangle pointing upstream. The flow is controlled by the independent rotation of the cylinders along their axis. The rotation of the cylinders enables the steering of incoming fluid particles, like a pinball machine. Thus, we refer this configuration as the fluidic pinball. In our study, we choose the side length of the equilateral triangle equal to be $1.5D$. Various side lengths have been explored numerically in (Chen et al., 2020), revealing a myriad of interesting regimes.

The flow is described in a Cartesian coordinate system, where the origin is located midway between the two rearward cylinders. The x-axis is parallel to the stream-wise direction. The y-axis is orthogonal to the cylinder axis. The velocity field is denoted by $\boldsymbol{u} = (u, v)$ and the pressure field by p. Here, u and v are, respectively, the stream-wise and transverse components of the velocity. We consider a Newtonian fluid of constant density ρ and kinematic viscosity ν. For the direct numerical simulation, the unsteady incompressible viscous Navier-Stokes equations are non-dimensionalized with cylinder diameter D, the incoming velocity U_∞ and the fluid density ρ. The corresponding Reynolds number is $\mathrm{Re}_D = U_\infty D/\nu$. Throughout this study, only $\mathrm{Re}_D = 100$ is considered.

The computational domain Ω is a rectangle bounded by $[-6, 20] \times [-6, 6]$ excluding the interior of the cylinders:

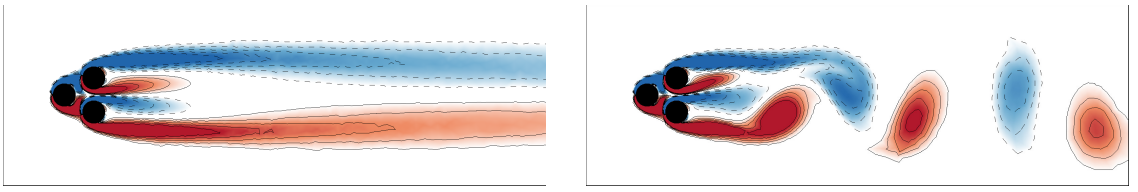
$$\Omega = \{[x, y]^\mathrm{T} \in \mathcal{R}^2 \colon [x, y]^\mathrm{T} \in [-6, 20] \times [-6, 6] \wedge (x - x_i)^2 + (y - y_i)^2 \geq 1/4, i = 1, 2, 3\}.$$

Here, $[x_i, y_i]^\mathrm{T}$ with $i = 1, 2, 3$, are the coordinates of the cylinder centers, starting from the front cylinder and numbered in mathematically positive direction,

$$\begin{array}{llll} x_1 = & -3/2\cos(30^\circ) & y_1 = & 0, \\ x_2 = & 0 & y_2 = & -3/4, \\ x_3 = & 0 & y_3 = & 3/4. \end{array}$$

The computational domain Ω is discretized on an unstructured grid comprising 4225 triangles and 8633 nodes. The grid is optimized to provide a balance between computation speed and accuracy. Grid independence of the direct Navier-Stokes solutions has been established by Deng et al. (2020).

The boundary conditions for the inflow, upper and lower boundaries are $U_\infty = \boldsymbol{e}_x$ while a stress-free condition is assumed for the outflow boundary. The control of the fluidic pinball is carried out by the rotation of the cylinders. A non-slip condition is adopted on the cylinders: the flow adopts the circumferential velocities of the front, bottom and top cylinder specified by $b_1 = U_F$, $b_2 = U_B$ and $b_3 = U_T$. The actuation command comprises these velocities, $\boldsymbol{b} = [b_1, b_2, b_3]^{\mathrm{T}}$. A positive (negative) value of the actuation command corresponds to counter-clockwise (clockwise) rotation of the cylinders along their axis. The numerical integration of the Navier-Stokes equations is carried by an in-house solver using a fully implicit Finite-Element Method (Noack et al., 2003, 2016). The method is third order accurate in time and space. The initial condition for the numerical simulations is the symmetric



(a) Symmetric steady solution.

(b) Unforced flow at $t = 400$.

Figure 4.1: Vorticity fields for the unforced fluidic pinball at $\mathrm{Re}_D = 100$. Blue (red) regions bounded by dashed lines represent negative (positive) vorticity. Darker regions indicate higher values of vorticity magnitude. Figures reprinted from Cornejo Maceda et al. (2021).

steady solution, see figure 4.1a. The symmetrical steady solution is computed with a Newton-Raphson method on the steady Navier-Stokes. An initial short, small rotation of the front cylinder is used to kick-start the transient to natural vortex shedding in the first period (Deng et al., 2020). The transient regime lasts around 400 convective time units. Figure 4.1 shows the vorticity field for the symmetric steady solution and the natural unforced flow after 400 convective units. The snapshot at $t = 400$ in figure 4.1b is the initial condition for all the following simulations.

4.1.2 Unforced reference

The fluidic pinball is a geometrically simple configuration that comprises key features of real-life flows such as successive bifurcations and frequency crosstalk between modes. Deng et al. (2020) shows that the unforced fluidic pinball undergoes successive bifurcations with increasing Reynolds number before reaching a chaotic regime. The first Hopf bifurcation at Reynolds number $\mathrm{Re} \approx 18$ breaks the symmetry in the flow and initiates the von Kármán vortex shedding. The second bifurcation at Reynolds number $\mathrm{Re} \approx 68$ is of pitchfork type and gives rise to a transverse deflection of jet-like flow appearing between the two rearward cylinders. The bi-stability of the jet deflection has been reported by Deng et al. (2020). At a Reynolds number $\mathrm{Re} = 100$ the jet deflection is rapid and occurs before the vortex shedding is fully established. Figure 4.2a shows an increase of the lift coefficient C_L before oscillations set in and the lift coefficient converges against a periodic oscillation around a slightly reduced mean value. Those bifurcations are a consequence of multiple instabilities present in the flow: there are two shear instabilities, on the top and bottom cylinder and a jet bi-stability originating from the gap between the two back cylinders. The shear-layer instabilities synchronize to a von Kármán vortex shedding. Figure 4.2 illustrates the dynamics of the unforced flow from the unstable steady symmetric solution to the post-transient periodic flow. The phase portrait in figure 4.2b and the power spectral density (PSD) in figure 4.2d show a periodic regime with frequency $f_0 = 0.116$ and its harmonic. Figure 4.2a shows that the mean value of the lift coefficient C_L is not null. This is due to the deflection of the jet behind the two rearward cylinders during the post-transient regime. During this regime, the deflection of the jet stays on one side as it is illustrated in figure 4.3a-4.3h over one period and in figure 4.3j in the mean field. This deflection explains the lift coefficient C_L asymmetry. Indeed, the upward oriented jet increases the pressure on the lower part of the top cylinder leading to an increase of the lift coefficient. In figure 4.2a, the initial downward spike on the lift coefficient is due to the initial kick. The unforced natural flow is our reference simulation for future comparisons.

Thanks to the rotation of the cylinders, the fluidic pinball is capable of reproducing six actuation

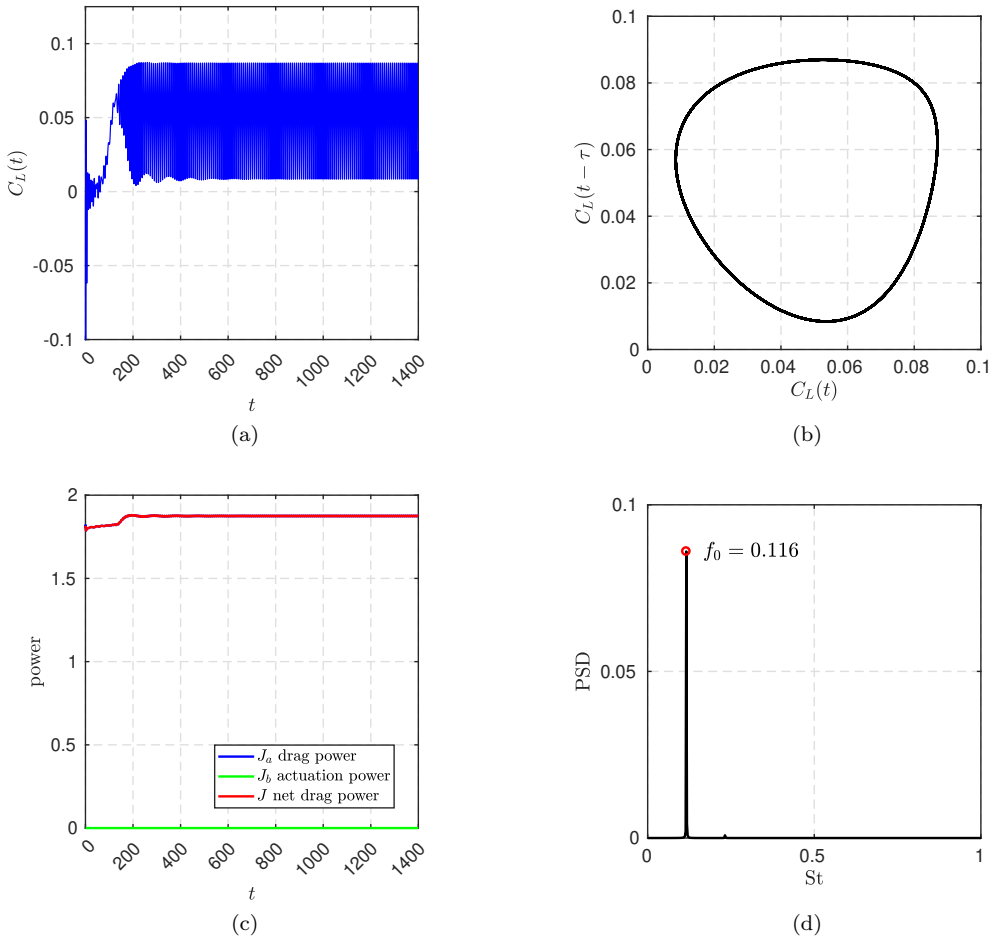
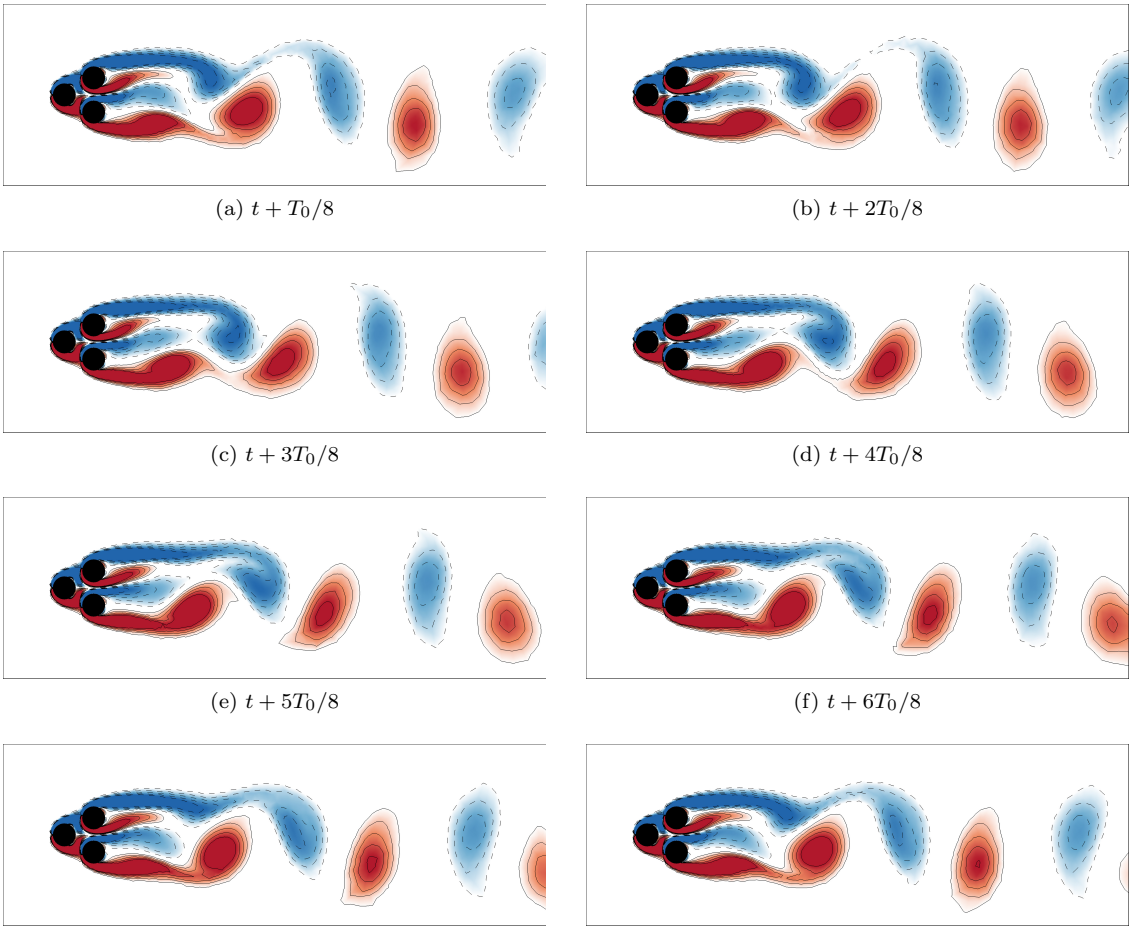


Figure 4.2: Characteristics of the unforced natural flow starting from the steady solution ($t = 0$). The transient spans until $t \approx 400$. (a) Time evolution of the lift coefficient C_L, (b) phase portrait, (c) time evolution of the drag power J_a (blue), actuation power J_b (green) and net drag power J (red) and (d) Power Spectral Density (PSD) showing the natural frequency $f_0 = 0.116$. The phase portrait is computed during the post-transient regime $t \in [900, 1400]$ and the PSD is computed over the last 1000 convective time units, $t \in [400, 1400]$.

mechanisms inspired from wake stabilization literature and exploiting distinct physics. Examples of those mechanisms can be found in Ishar et al. (2019). First, the wake can be stabilized by shaping the wake region more aerodynamically—also called fluidic boat tailing. The shear layers are vectored towards the center region with passive devices, like vanes (Flügel, 1930) or active control through Coanda blowing (Geropp, 1995; Geropp and Odenthal, 2000; Barros et al., 2016). In the case of the fluidic pinball, we can mimic this effect by a counter-rotating rearward cylinders which accelerates the boundary layers and delays separation. This fluidic boat tailing is typically associated with significant drag reduction. Second, the two rearward cylinders can also rotate oppositely ejecting a fluid jet on the centerline. Thus, interaction between the upper and lower shear layer is suppressed, preventing the development of a von Kármán vortex in the vicinity of the cylinders. Such base bleeding mechanisms has a similar physical effect as a splitter plate behind a bluff body and has been proved to be an effective means for wake stabilization (Wood, 1964; Bearman, 1967).

Third, phasor control can be performed by estimating the oscillation phase and feeding it back with a phase shift and gain (Protas, 2004). Fourth, unified rotation of the three cylinders in the same direction gives rise to higher velocities, and thus larger vorticity, on one side at the expense of the other side, destroying the vortex shedding. This effect relates to the Magnus effect and stagnation point control (Seifert, 2012). Fifth, high-frequency forcing can be effected by symmetric periodic oscillation of the rearward cylinders. With a vigorous cylinder rotation (Thiria et al., 2006), the upper and lower shear layers are re-energized, reducing the transverse wake profile gradients and thus the instability of the flow. Thus, the effective eddy viscosity in the von Kármán vortices increases, adding a damping effect. Sixth



(a) $t + T_0/8$ (b) $t + 2T_0/8$

(c) $t + 3T_0/8$ (d) $t + 4T_0/8$

(e) $t + 5T_0/8$ (f) $t + 6T_0/8$

(g) $t + 7T_0/8$ (h) $t + T_0$

Figure 4.3: Vorticity fields of the unforced flow. (a)-(f) Time evolution of the vorticity field in the last period of the simulation. The color code is the same as figure 4.1. T_0 is the natural period associated to the natural frequency f_0.

and finally, a symmetrical forcing at a lower frequency than the natural vortex shedding may stabilize the wake (Pastoor et al., 2008). This is due to the mismatch between the anti-symmetric vortex shedding and the forced symmetric dynamics whose clock-work is distinctly out of sync with the shedding period. High- and low-frequency forcing lead to frequency crosstalk between actuation and vortex shedding over the mean flows, as described by low-dimensional generalized mean-field model (Luchtenburg et al., 2009).

We confirm therefore that the fluidic pinball is an interesting Multiple-Input Multiple-Output (MIMO) control benchmark. The configuration exhibits well-known wake stabilization mechanisms in physics. From a dynamical perspective, nonlinear frequency crosstalk can easily be enforced. In addition, even long-term simulations can easily be performed on a laptop within an hour.

4.1.3 Control objective and regression problem

Several control objectives related to the suppression or reduction of undesired forces can be considered for the fluidic pinball. We can increase the recirculation bubble length, reduce lift fluctuations or even mitigate the total fluctuation energy.

In this study, we aim to reduce the net drag power at $\mathrm{Re}_D = 100$. The associated objectives are J_a, the drag power and J_b, the actuation power. The cost J_a is defined as the temporal average of the drag power of the controlled flow field:

$$J_a = \frac{1}{T_{ev}} \int_{t_0}^{t_0+T_{ev}} j_a(t)\, \mathrm{d}t \tag{4.1}$$

with the instantaneous cost function

$$j_a(t) = \boldsymbol{F}_x(t) \cdot \boldsymbol{U}_\infty \tag{4.2}$$

where $\boldsymbol{F}_x$ is the drag and $\boldsymbol{U}_\infty$ is the incoming velocity. The control is activated at $t_0 = 400$ convective time units after the starting kick on the steady solution. Thus, we have a fully established post-transient regime. The cost function is evaluated until $T_{\text{ev}} = 525$ convective time units. Thus, the time average is effected over 125 convective time units which corresponds to more than 10 periods T_0 of the unforced flow.

J_b is naturally chosen as a measurement of the actuation energy investment. Evidently, a low actuation energy is desirable. The actuation power is computed as the power of the torque applied by the fluid on the cylinders. J_b is the time-averaged actuation power over $T_{\text{ev}} = 125$ time units:

$$J_b(\boldsymbol{b}) = \frac{1}{T_{ev}} \int_{t_0}^{t_0+T_{\text{ev}}} \sum_{i=1}^{3} \mathcal{P}_{\text{act,i}} \, \mathrm{d}t \tag{4.3}$$

where $\mathcal{P}_{\text{act,i}}$ is the actuation power supplied integrated over cylinder i:

$$\mathcal{P}_{\text{act,i}} = - \oiint b_i F^{\theta}_{s,i} \, \mathrm{d}s$$

where $\left(F^{\theta}_{s,i}\mathrm{d}s\right)$ is the azimuthal component of the local fluid forces applied to cylinder i. The negative sign denotes that the power is supplied and not received by the cylinders.

Thus, the cost function employed for the optimization is $J = J_a + \gamma J_b$. γ is the penalization parameter. It allows to balance the terms of the cost function. In this study as we aim to reduce the net drag power, we set $\gamma = 1$ so both components J_a and J_b have the same weight.

The instantaneous values of J_a and J_b are plotted in figure 4.2c. Naturally, for the unforced flow, the actuation power is null, and the cost function is only the drag power. We note that the drag power takes around 300 convective units to stabilize. The cost of the post-transient regime is $J_0 = 1.87$. J_0 serves as a reference for future comparisons. The cost of the steady flow 4.1a is $J_{\text{steady}} = 1.80$ which is lower than J_0 but still high. Therefore, we can assume that stabilizing the symmetric steady solution may not be the best strategy to reduce the net drag power.

In order to minimize the net drag power, the flow is forced by the rotation of the three cylinders. The actuation command $\boldsymbol{b} = [b_1, b_2, b_3]^{\mathrm{T}}$ is determined by the control law $\boldsymbol{K}$. This control law may operate open-loop or closed-loop with flow input. Considered open-loop actuations are steady or harmonic oscillation around a vanishing mean. Considered feedback includes velocity sensor signals in the wake. Thus, in the most general formulation, the control law reads the equation described in Sec. 2.1: $\boldsymbol{b}(t) = \boldsymbol{K}(\boldsymbol{h}(t), \boldsymbol{s}(t))$ with $\boldsymbol{h}(t)$ and $\boldsymbol{s}(t)$ being vectors comprising respectively time dependent harmonic functions and sensor signals. The sensor signals include the instantaneous velocity signals as well as three recorded values over one period as elaborated in the result (Sec. 4.4). In the following, N_{b} represents the number of actuators, N_{H} for the number of time-dependent functions and N_{S} for the number of sensor signals.

4.2 Symmetric steady actuation for net drag reduction

First, we carry out an open-loop parametric study to assess the effect of control on the drag power. To achieve an exhaustive parametric study is a costly task as we need to operate in 3-dimensional parameter space. That is why we restrict the search to the subspace of symmetric actuations: the front cylinder does not rotate and the two back cylinders rotate at the same speed but in opposite directions:

$$\begin{aligned} b_1 &= 0, \\ b_2 &= -b_3. \end{aligned}$$

Thus, we explore the effect of only one parameter b_2. b_2 is defined so that when it is positive, the flow is vectored towards the centerline—boat tailing configuration—and when it is negative, the inner flow is accelerated—base bleeding configuration. Figure 4.4 shows the evolution of J_a, J_b and $J = J_a + J_b$ as a function of b_2. Solely considering J_a, boat tailing is the best strategy to reduce the drag power. Indeed, drag power decreases monotonously with increasing b_2. For a strong actuation, $b_2 > 3$, J_a even becomes negative and the fluidic pinball becomes a jet. Base bleeding, on the other hand, is not a viable strategy to reduce the drag power. There is a local minimum around $b_2 = -4$ but its associated drag power is still higher than the natural unforced flow. When adding the actuation power, there is only one minimum for the cost function J, around $b_2 = 1$. Its associated cost is $J_{\text{BT}}/J_0 = 0.77$. Figure 4.5d shows the characteristics of the controlled flow with the best boat tailing solution. We note that the regime is purely harmonic with an increase of the main frequency from $f_0 = 0.116$, for the unforced flow, to

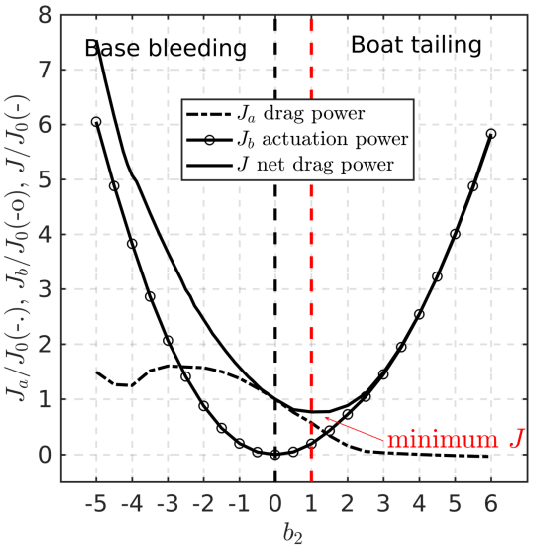


Figure 4.4: Cost function evolution regarding actuation intensity for symmetric constant actuation.

$f_1 = 0.128$. We also remark a symmetrization of the lift coefficient C_L alongside a significant increase of the oscillations amplitude. The loss of the mean value results in a symmetrization of the flow. Indeed, figure 4.6 shows that the near jet completely disappeared. We also observe a considerable reduction of the recirculation bubble, as a consequence, the base pressure behind the back cylinders increases and the total drag decreases.

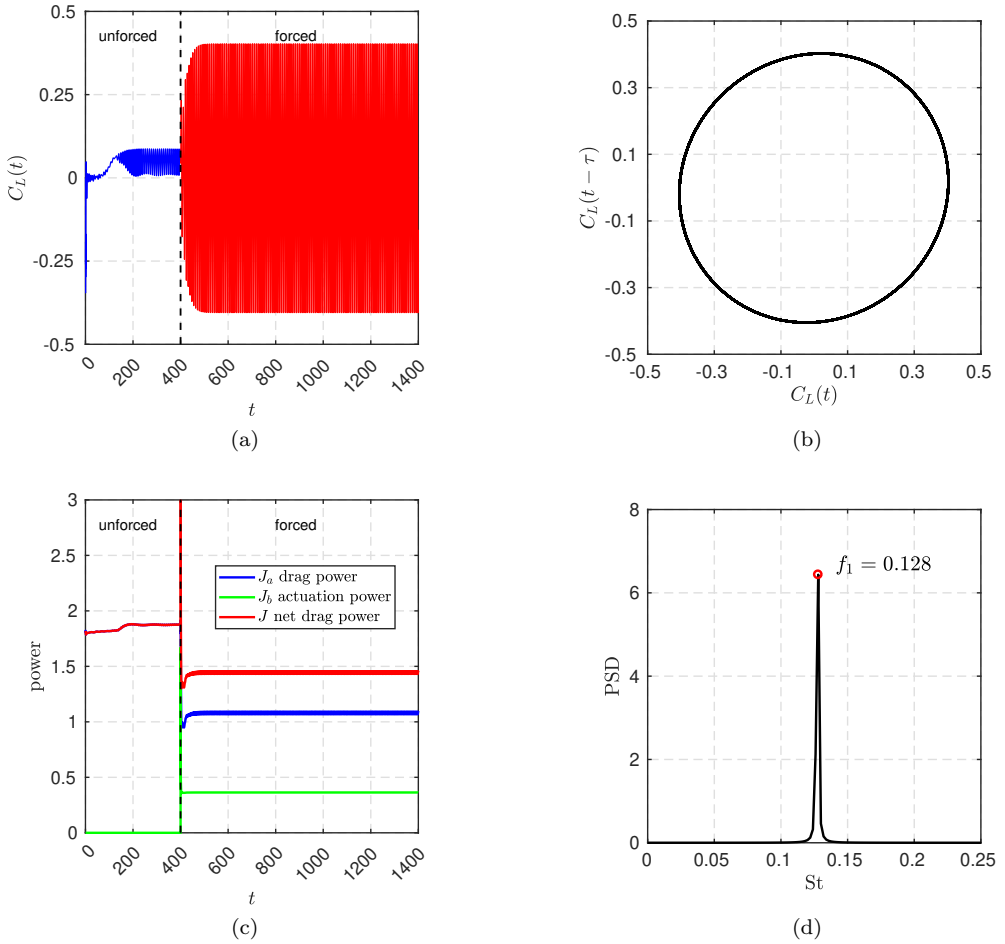
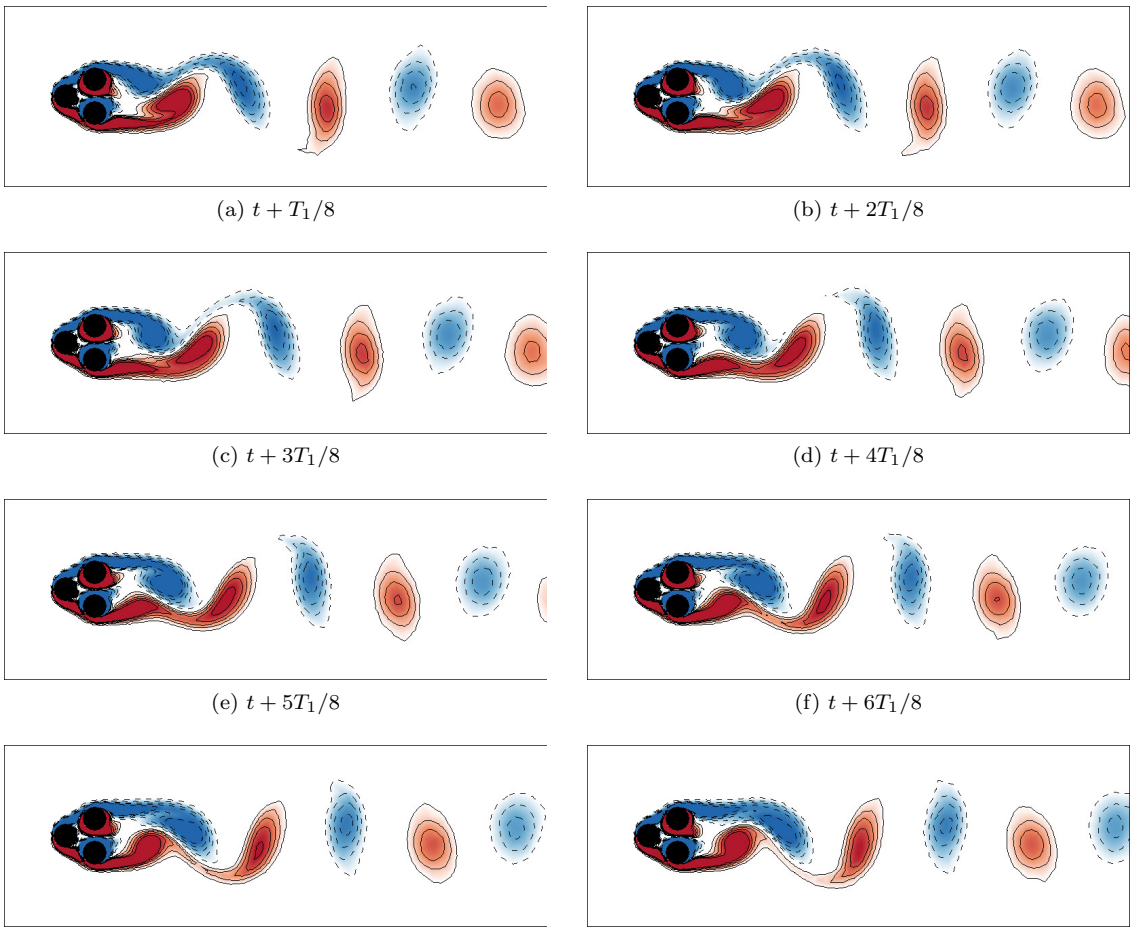


Figure 4.5: Characteristics of the best boat tailing solution starting from the steady solution ($t = 0$). The transient spans until $t \approx 400$. (a) Time evolution of the lift coefficient C_L, (b) phase portrait, (c) time evolution of the drag power J_a (blue), actuation power J_b (green) and net drag power J (red) and (d) Power Spectral Density (PSD) showing the frequency $f_1 = 0.128$.



(a) $t + T_1/8$ (b) $t + 2T_1/8$

(c) $t + 3T_1/8$ (d) $t + 4T_1/8$

(e) $t + 5T_1/8$ (f) $t + 6T_1/8$

(g) $t + 7T_1/8$ (h) $t + T_1$

Figure 4.6: Vorticity fields of the flow controlled by the best boat tailing solution. (a)-(f) Time evolution of the vorticity field in the last period of the 1000 time units simulation. The color code is the same as figure 4.1. T_1 is the period associated to the frequency f_1.

4.3 Multi-frequency optimization

In this section, we explore the space of open-loop controllers, in particular periodic control laws. In order to do so, we consider cosine functions as inputs for the control laws. Thus, equation (2.1) becomes: $\boldsymbol{b}(t) = \boldsymbol{K}(\boldsymbol{h}(t))$. To enrich our search space and to avoid resonance effects, we choose eight periodic functions whose frequencies are incommensurable with the natural frequency f_0:

$$\begin{aligned} h_1 &= \cos(2\pi\Phi^{-4}f_0t), & h_5 &= \cos(2\pi\Phi^{1}f_0t),\\ h_2 &= \cos(2\pi\Phi^{-3}f_0t), & h_6 &= \cos(2\pi\Phi^{2}f_0t),\\ h_3 &= \cos(2\pi\Phi^{-2}f_0t), & h_7 &= \cos(2\pi\Phi^{3}f_0t),\\ h_4 &= \cos(2\pi\Phi^{-1}f_0t), & h_8 &= \cos(2\pi\Phi^{4}f_0t). \end{aligned}$$

The golden ration Φ assures that the periodic function is incommensurable with the natural frequency f_0, i.e. the natural frequency cannot be reconstructed thanks to the algebraic operators $(+, -, \times, \div)$. The rest of the MLC parameters are summarized in table 4.1. We choose the operators probability

Parameter	Description	Value
	Function library	$F_2 = \{+, -, \times, \div, \exp, \tanh, \sin, \cos, \log\}$
$\boldsymbol{h}$	Controller inputs	$h_i,\ i = -4, .., -1, 1, ..4$
N_{Var}	Number of variable registers	$8 + 3 = 11$
N_{Cst}	Number of constant registers	10
$N_{\text{Instr,max}}$	Max. number of instructions	50
$N_{\text{Pop.size}}$	Population size	100
N_{G}	Number of generations	10
N_{Tour}	Tournament size	7
N_{E}	Elitism	1
P_{Cros}	Crossover probability	0.6
P_{Mut}	Mutation probability	0.3
P_{Rep}	Replication probability	0.1

Table 4.1: MLC parameters for multi-frequency forcing optimization.

$(P_{\text{Cros}}, P_{\text{Mut}}, P_{\text{Rep}}) = (0.6, 0.3, 0.1)$ as explained in App. A.2. As we optimize three controllers, we decided to increase the number of maximum instructions to 50. To build complex control laws, we employ the function library $F_2 = \{+, -, \times, \div, \exp, \tanh, \sin, \cos, \log\}$. Also, since we have eight inputs for the control laws, we need to increase the number of variable registers to include an instance of all inputs. Thus, we also increase the number of constants. We ran the optimization with a population of 100 individuals evolving through 10 generations. The learning process is illustrated in figure 4.7. Most of the learning is done during the Monte Carlo step, indeed after 100 random evaluations the cost is equal to $J/J_0 = 0.80$. Only small improvements are made until the fourth generation where a big jump manages to reduce the cost lower than the boat tailing solution. From there, only small improvements are achieved.

The best control law $\boldsymbol{b}^{\text{MF}}$ reads:

$$\begin{aligned} b_1^{\text{MF}} &= -0.13732,\\ b_2^{\text{MF}} &= 0.982511,\\ b_3^{\text{MF}} &= -1.1979,\\ J_{\text{MF}}/J_0 &= 0.7476 \end{aligned} \tag{4.4}$$

We note that the control is steady and does not contain any h_i, suggesting that periodic forcing is not a viable solution to reduce the net drag power. Indeed, periodic forcing must increase the gradient of the azimuthal speed, thus increasing the torque and actuation power. $\boldsymbol{b}^{\text{MF}}$ resembles a boat tailing configuration with a slight asymmetry as the bottom cylinders rotates faster than the top one and the front cylinder also presents a small rotation. We suspect that this asymmetry is typical of pitchfork bifurcated flows as it was also reported by Raibaudo et al. (2020). The characteristics of the flow controlled by $\boldsymbol{b}^{\text{MF}}$ are shown in figure 4.8. As for the best boat tailing solution, the controlled flow with $\boldsymbol{b}^{\text{MF}}$ is purely harmonic with a slightly lower frequency $f_2 = 0.126$. The amplitude of the oscillations of the lift coefficient has also increased. However, the slight asymmetry in the control results in a significant increase of the mean value of the lift coefficient. The mean value variation of the lift coefficient is,

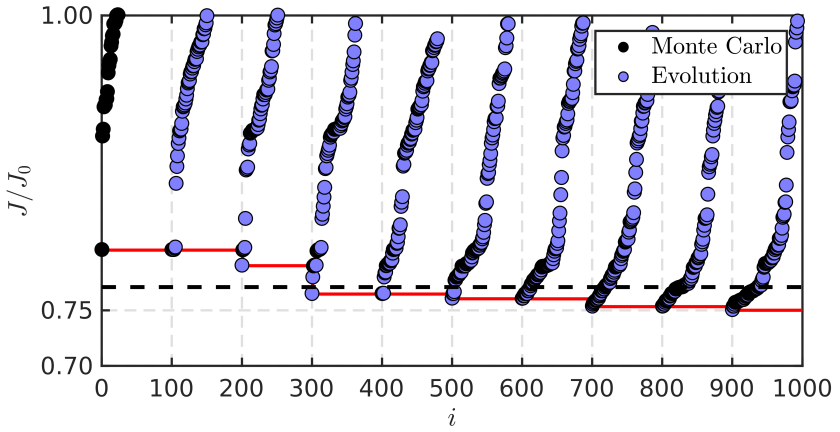


Figure 4.7: Distribution of the costs during the multi-frequency forcing MLC optimization process. Each dot represents the cost J_a/J_0 of one individual. The color of the dots represents how the individuals have been generated. Black dots denote the individuals which are randomly generated (Monte Carlo). Blue dots refer to individuals which are generated from a genetic operator. The individuals from each generation have been sorted following their costs. The red line shows the evolution of the best cost. The dashed horizontal line corresponds to the best symmetric solution $b_2 = 1$ with a cost of $J_{\mathrm{BT}}/J_0 = 0.77$. The vertical axis is in log scale and has been truncated to help the visualization of the best individuals.

nonetheless, hardly visible on the snapshots in figure 4.9. As the actuation is close to the best boat tailing control, the controlled flows are also similar. By squinting ones eye, we notice that there is a small region, at the lower-back part of the front cylinder, with intense vorticity. We can assume that this small vortex increased the local pressure and thus shifts the mean value of the lift positively.

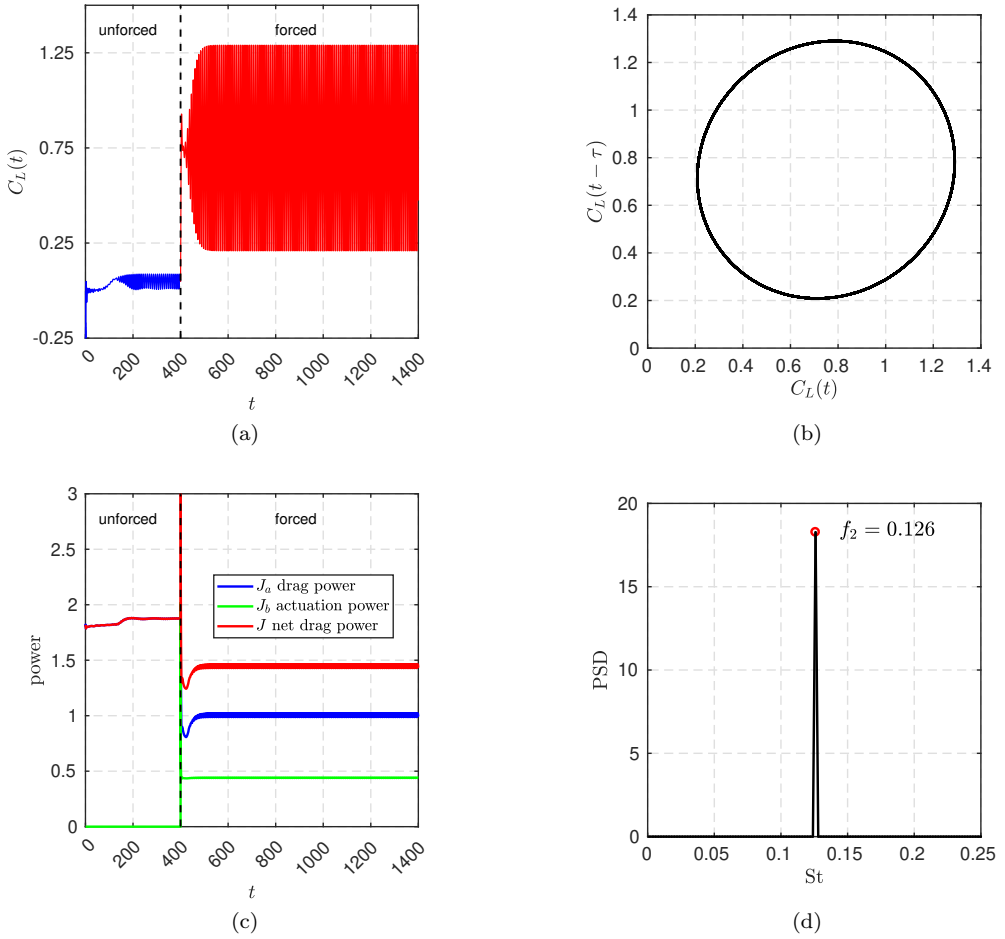
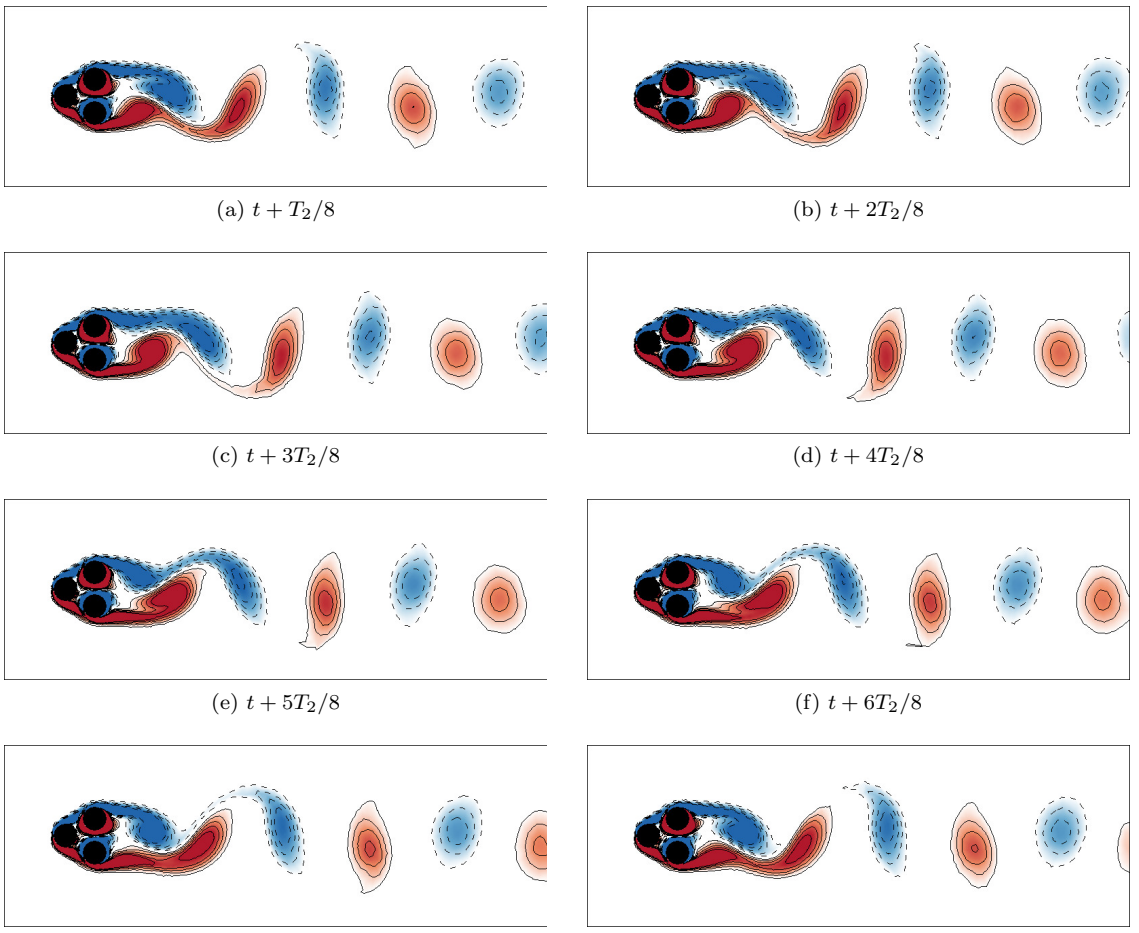


Figure 4.8: Characteristics of the flow controlled by $\boldsymbol{b}^{\mathrm{MF}}$ starting from the steady solution ($t = 0$). The transient spans until $t \approx 400$. (a) Time evolution of the lift coefficient C_L, (b) phase portrait, (c) time evolution of the drag power J_a (blue), actuation power J_b (green) and net drag power J (red) and (d) Power Spectral Density (PSD) showing the frequency $f_2 = 0.126$.



(a) $t + T_2/8$ (b) $t + 2T_2/8$

(c) $t + 3T_2/8$ (d) $t + 4T_2/8$

(e) $t + 5T_2/8$ (f) $t + 6T_2/8$

(g) $t + 7T_2/8$ (h) $t + T_2$

Figure 4.9: Vorticity fields of the flow controlled by the best boat tailing solution. (a)-(f) Time evolution of the vorticity field in the last period of the 1000 time units simulation. The color code is the same as figure 4.1. T_2 is the period associated to the frequency f_2.

4.4 Feedback control law optimization

In this section, we allow feedback control laws by adding sensor signals as inputs. Thus, equation (2.1) becomes: $\boldsymbol{b}(t) = \boldsymbol{K}(\boldsymbol{s}(t))$. We choose a grid of nine sensor downstream measuring either x or y velocity components. The coordinates of the sensors are $x = 5,\ 6.5,\ 8$ and $y = 1.25,\ 0,\ -1.25$. The six exterior sensors are u sensors while v sensors are chosen for the ones on the symmetry line $y = 0$. The information

Sensor	x-coordinate	y-coordinate	Velocity component
s_1	5	1.25	u
s_2	6.5	1.25	u
s_3	8	1.25	u
s_4	5	0	v
s_5	6.5	0	v
s_6	8	0	v
s_7	5	-1.25	u
s_8	6.5	-1.25	u
s_9	8	-1.25	u

Table 4.2: Summary of sensor information.

of sensors is summarized in table 4.2. Moreover, in order to take into account the convective nature of the flow, we add time-delayed sensors as inputs of the control laws. The delays are a quarter, half and three-quarters of the unforced natural period, yielding following additional lifted sensor signals:

$$s_{i+9}(t) = s_i(t - T_0/4), \quad s_{i+18}(t) = s_i(t - T_0/2), \quad s_{i+27}(t) = s_i(t - 3T_0/4).$$

Hence, the dimension of the sensor vector $\boldsymbol{s}$ is $9 \times 4 = 36$ and $X \subset \mathbb{R}^{36}$. The time-delayed sensors are also include as control inputs to mimic ARMAX-based control (Hervé et al., 2012). The same MLC

Parameter	Description	Value
	Function library	$F_2 = \{+, -, \times, \div,$ $\exp, \tanh, \sin, \cos, \log\}$
$\boldsymbol{s}$	Controller inputs	$s_i(t),\ i = 1, .., 36$
N_{Var}	Number of variable registers	$36 + 3 = 39$
N_{Cst}	Number of constant registers	10
$N_{\text{Instr,max}}$	Max. number of instructions	50
$N_{\text{Pop.size}}$	Population size	100
N_{G}	Number of generations	10
N_{Tour}	Tournament size	7
N_{E}	Elitism	1
P_{Cros}	Crossover probability	0.6
P_{Mut}	Mutation probability	0.3
P_{Rep}	Replication probability	0.1

Table 4.3: MLC parameters for feedback control optimization.

parameters as for Sec. 4.3 have been chosen. The number of variable constants increased as we now have 36 sensor signals. Figure 4.10 shows cost of the individuals during the optimization process. We note the same learning trend as for multi-frequency forcing optimization. However in this case, the big jump appeared sooner, directly at the second generation. From there, only small improvements are achieved. We note also that the Monte Carlo step is less efficient as less individuals have reached a performance lower than the unforced one. Moreover the best individual of the first generation has a cost of $J/J_0 = 0.84$ which is higher than in the multi-frequency forcing case. This can be explained by the fact that as there are more inputs, the search space becomes larger thus the drop in performance of Monte Carlo.

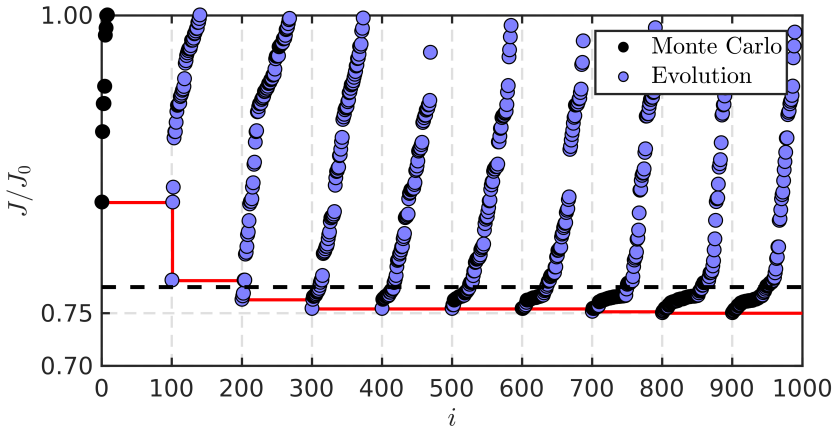


Figure 4.10: Same as figure 4.7 but for the feedback optimization MLC.

The expression of the best control law $\boldsymbol{b}^{\mathrm{FB}}$ is:

$$\begin{aligned} b_1^{\mathrm{FB}}(t) &= \log(s_9(t - T_0/2)), \\ b_2^{\mathrm{FB}}(t) &= \exp(0.18549\ \frac{s_1(t - 4T_0/3)}{s_9(t)}), \\ b_3^{\mathrm{FB}}(t) &= -1.12724, \\ J_{\mathrm{FB}}/J_0 &= 0.7451. \end{aligned} \tag{4.5}$$

MLC managed to successfully combined sensors signals, delayed sensor signals and nonlinear function to build a controller b^{FB} that reduces even further the cost function compared to the b^{MF} control law. The actuation command resulting from b^{FB} is plotted in figure 4.11. We note that the control resembles,

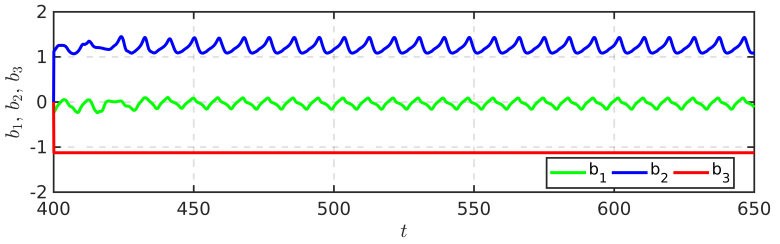


Figure 4.11: Time series of the actuation command for the best feedback control law found with MLC.

again, the boat tailing solution, however this strategy is augmented by a phasor control for the front and bottom cylinder, meaning that the control of the front and bottom cylinder is directly related to the oscillatory dynamics of the flow (Brunton and Noack, 2015). A spectral analysis shows that the main frequency of b_2 and b_3 are both $f_3 = 0.112$, suggesting that it is a direct feedback.

Figure 4.12 shows the characteristics of flow controlled by b^{FB}. The controlled flow is purely harmonic according to figure 4.12d, but the phase portrait, figure 4.12b, is slightly deformed due to the third harmonic (not shown in the figure), even though its amplitude is imperceptible. The amplitude oscillations of the lift coefficient also increased and the mean value is not null. The kinematics of the flow, figure 4.13, show that the near jet disappeared and the length of the recirculation decreased like the two last control strategies. However, we note that the recirculation bubble is a bit less reduced. Also, the vortices stay attached longer before shedding. This is especially true for the bottom part where the positive vortex stretches unusually, see figure 4.13c, 4.13d, 4.13e and 4.13f. The intensity of the positive vortices is also lesser than the previous control. This can be explained by the re-energization of the shear layers, especially the bottom one, due to the periodic component of the forcing, like Protas (2004). The rotation of the front cylinder has been reported in other studies, such as Cornejo Maceda et al. (2019), but its effect is not yet fully understood.

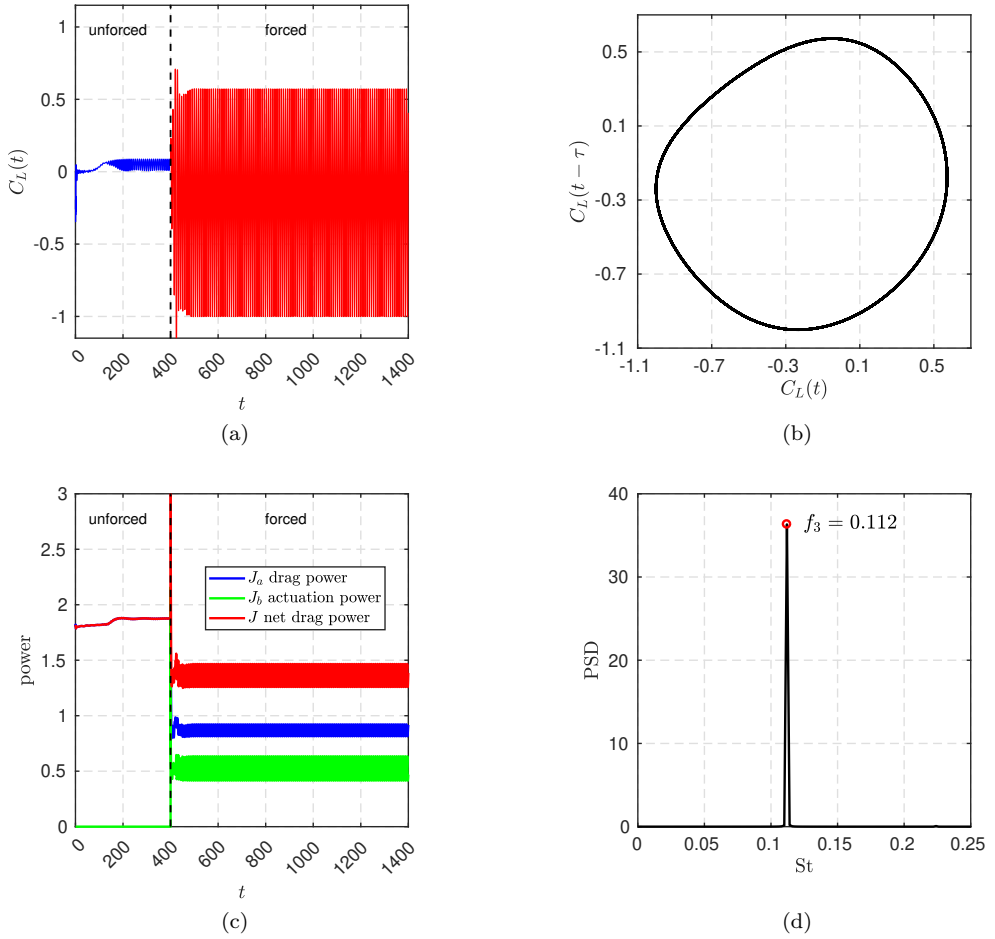
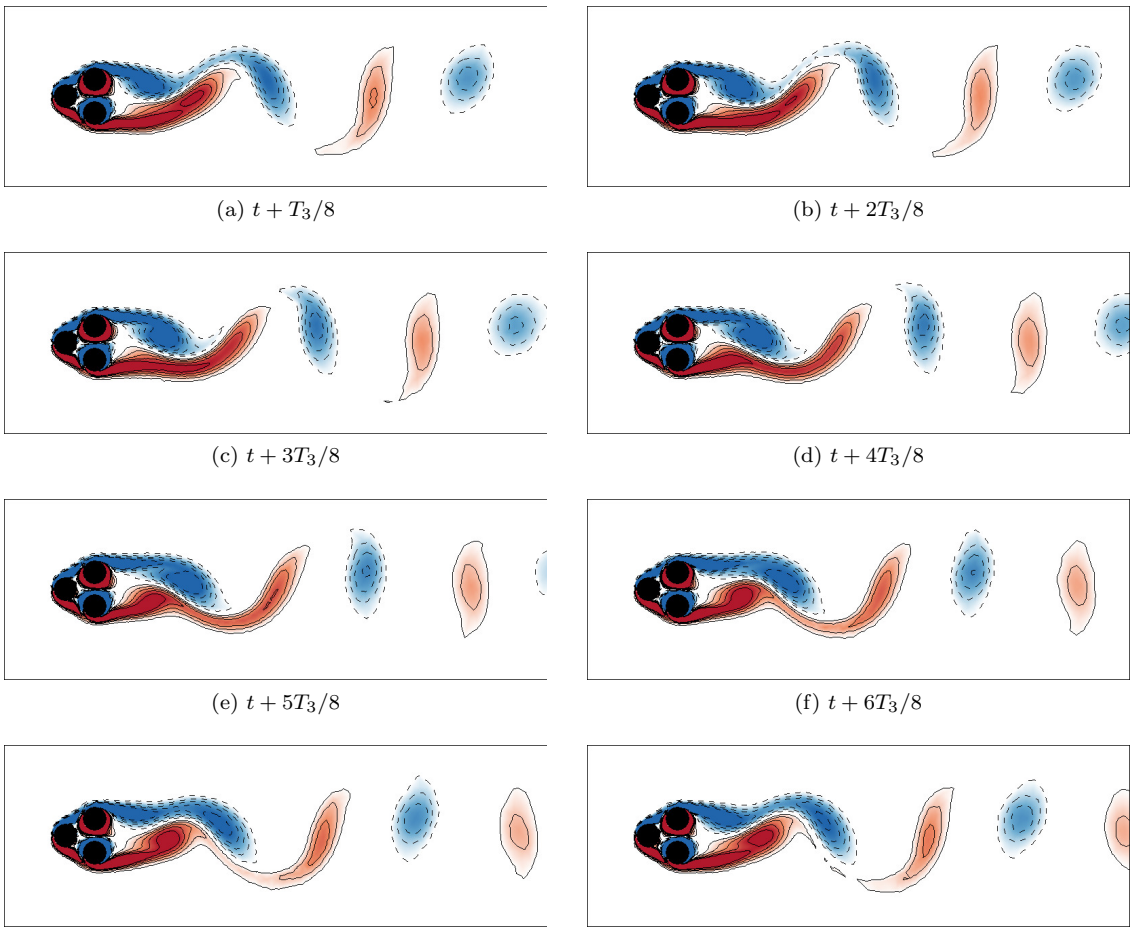


Figure 4.12: Characteristics of the flow controlled by $\boldsymbol{b}^{\mathrm{FB}}$ starting from the steady solution ($t = 0$). The transient spans until $t \approx 400$. (a) Time evolution of the lift coefficient C_L, (b) phase portrait, (c) time evolution of the drag power J_a (blue), actuation power J_b (green) and net drag power J (red) and (d) Power Spectral Density (PSD) showing the frequency $f_3 = 0.112$.



(a) $t + T_3/8$ (b) $t + 2T_3/8$

(c) $t + 3T_3/8$ (d) $t + 4T_3/8$

(e) $t + 5T_3/8$ (f) $t + 6T_3/8$

(g) $t + 7T_3/8$ (h) $t + T_3$

Figure 4.13: Vorticity fields of the flow controlled by the feedback control law derived by MLC (a)-(f) Time evolution of the vorticity field in the last period of the 1000 time units simulation. The color code is the same as figure 4.1. T_3 is the period associated to the frequency f_3.

4.5 General control law optimization: multi-frequency and feedback control

Parameter	Description	Value
	Function library	$F_2 = \{+, -, \times, \div,$ $\exp, \tanh, \sin, \cos, \log\}$
$\boldsymbol{s}$, $\boldsymbol{h}$	Controller inputs	$s_i(t)$, $i = 1..36$ $h_i(t)$, $i = -4, .., -1, 1, ..4$
N_{Var}	Number of variable registers	$8 + 36 + 3 = 47$
N_{Cst}	Number of constant registers	10
$N_{\text{Instr,max}}$	Max. number of instructions	50
$N_{\text{Pop.size}}$	Population size	100
N_{G}	Number of generations	10
N_{Tour}	Tournament size	7
N_{E}	Elitism	1
P_{Cros}	Crossover probability	0.6
P_{Mut}	Mutation probability	0.3
P_{Rep}	Replication probability	0.1

Table 4.4: MLC parameters for general optimization including multi-frequency forcing and feedback control.

In this last section, we run a hybrid optimization allowing both multi-frequency forcing and feedback control. We have seen in Sec. 4.3, that periodic forcing has not been selected to control the fluidic pinball, this may be related to the difficulty to build the proper frequency for control. By adding flow information, we can expect MLC to build an open-loop periodic forcing modulated by a sensor signal and thus enable a richer control. Such approach has been successfully employed to reduce the recirculation bubble of a back-ward facing step at Reynolds number Re = 31500 in Chovet et al. (2017). Then, we allow $\boldsymbol{s}$ and $\boldsymbol{h}$ as inputs of the controller and equation (2.1) becomes: $\boldsymbol{b}(t) = \boldsymbol{K}(\boldsymbol{s}(t), \boldsymbol{h}(t))$. The MLC parameters are summarized in table 4.4. Figure 4.14 illustrates the learning process for the hybrid optimization.

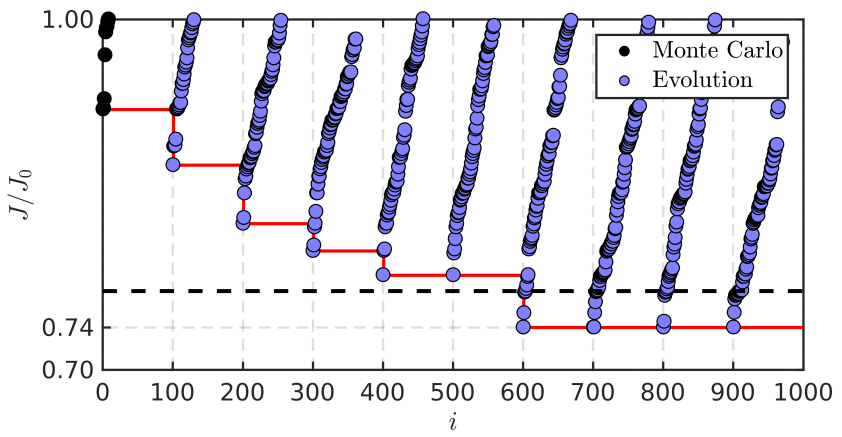


Figure 4.14: Same as figure 4.7 but for the hybrid optimization MLC.

First, Monte Carlo sampling struggles to find a good individual and after 100 random evaluations, the cost of the best individual is only $J/J_0 = 0.92$. As detailed in Sec. 4.4, this can be explained by the adding of extra inputs, which enlarges the search space. Contrary to the multi-frequency optimization and feedback control optimization, we note that significant and regular improvements are made at each generation until reaching a plateau at $J_{\text{HB}} = 0.7363$ after seven generations. For all generations the distribution of the individuals looks linear as opposed to the two previous optimizations where there was an accumulation of good individuals in the final generations. This may be explained by the fact that as new and more efficient individuals are built at each generation, the is still a lot of diversity in the population. When the learning is slowed down, then the best individual takes over the population

thanks to replication. From there, we enter a phase of fine tuning of the control law with only small improvements and thus an accumulation of good individuals in the generation. Such behavior is likely to be observed if more generations were computed.

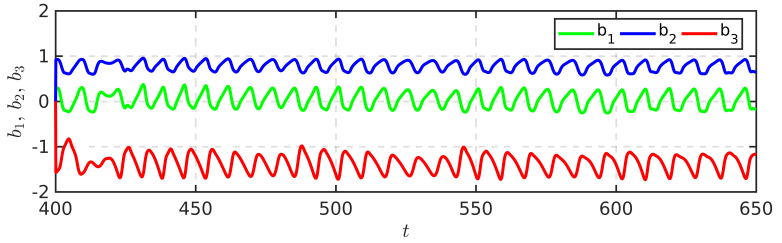


Figure 4.15: Time series of the actuation command for the best feedback control law found with MLC.

The final control law $\boldsymbol{b}^{\mathrm{HB}}$ reads:

$$\begin{array}{lcl} b_1^{\mathrm{HB}}(t) & = & \cos(\cos(s_8(t - T_0/2)) + 0.88123), \\ b_2^{\mathrm{HB}}(t) & = & \cos(\cos(s_8(t - T_0/2))), \\ b_3^{\mathrm{HB}}(t) & = & -0.36574 - s_2(t), \\ J_{\mathrm{HB}}/J_0 & = & 0.7363 \end{array} \tag{4.6}$$

The control built includes sensor information, a nonlinear function, cos, but no open-loop periodic function. The three components contain feedback information. The time series of this control are plotted in figure 4.15. The mean values of all controllers are in line with a boat tailing configuration, however there is a non-negligible oscillatory component for all actuations. The three cylinders are in direct feedback as the dominant frequency of the actuation commands are $f_4 = 0.108$, the main frequency of the flow. So far, b^{HB} is the control that reduces the most the cost function with $J_{\mathrm{HB}}/J_0 = 0.7363$. However, in figure 4.16a and 4.16c, we notice that the transient extends until ≈ 700 convective time units. The cost of the controlled flow computed on the post-transient regime is $J/J_0 = 0.7369$, showing that the extended transient only brings a negligible improvement. We note that taking into account a longer time-window may enable solutions with long transients. We notice in figure 4.16a that the oscillations of the lift coefficient increased alongside with its mean value. Yet, this asymmetry is not obvious in the figure 4.17. Controlled with b^{HB}, the flow is similar to the previous feedback control. We note, nevertheless, that the recirculation bubble is slightly larger. As for b^{FB}, the vortices stay attached longer thanks to the closed-loop periodic forcing. We notice in particular that the positive vortex is shed after a longer time interval than the negative vortex, indeed we notice a larger distance between a positive vortex and the previous negative vortex downstream than between a negative vortex and the previous positive vortex downstream.

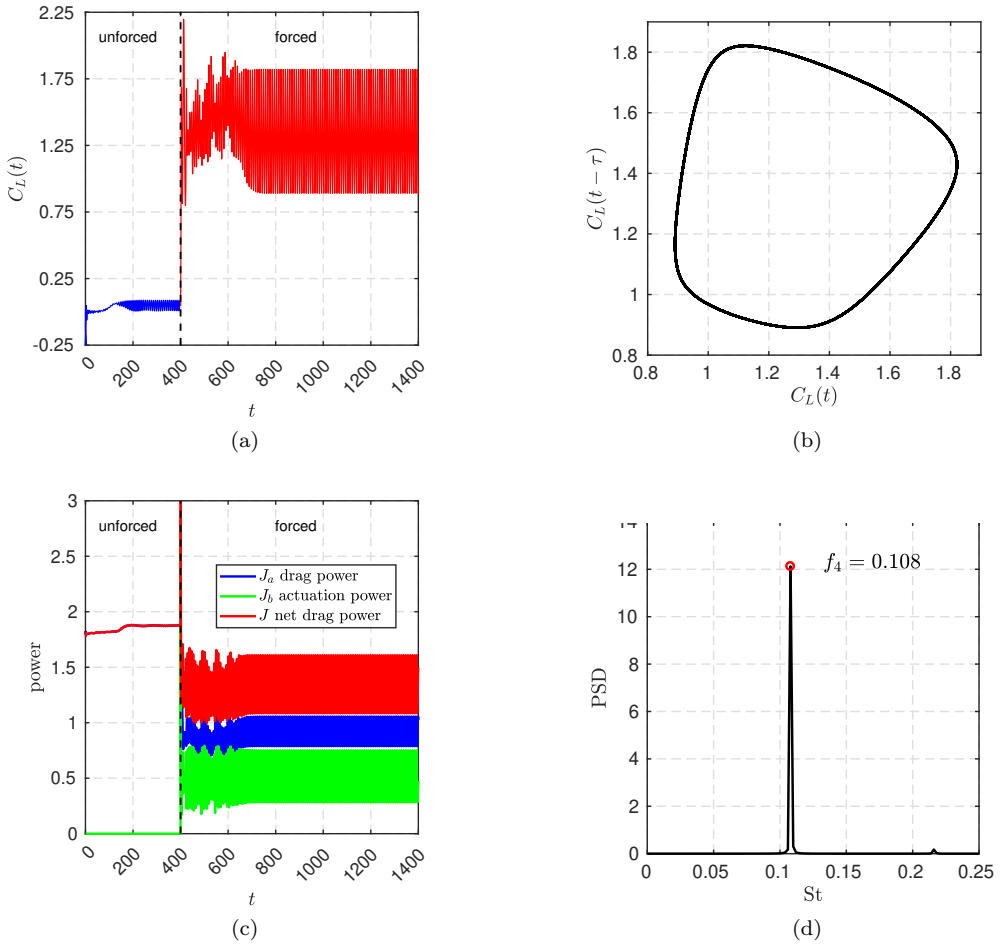
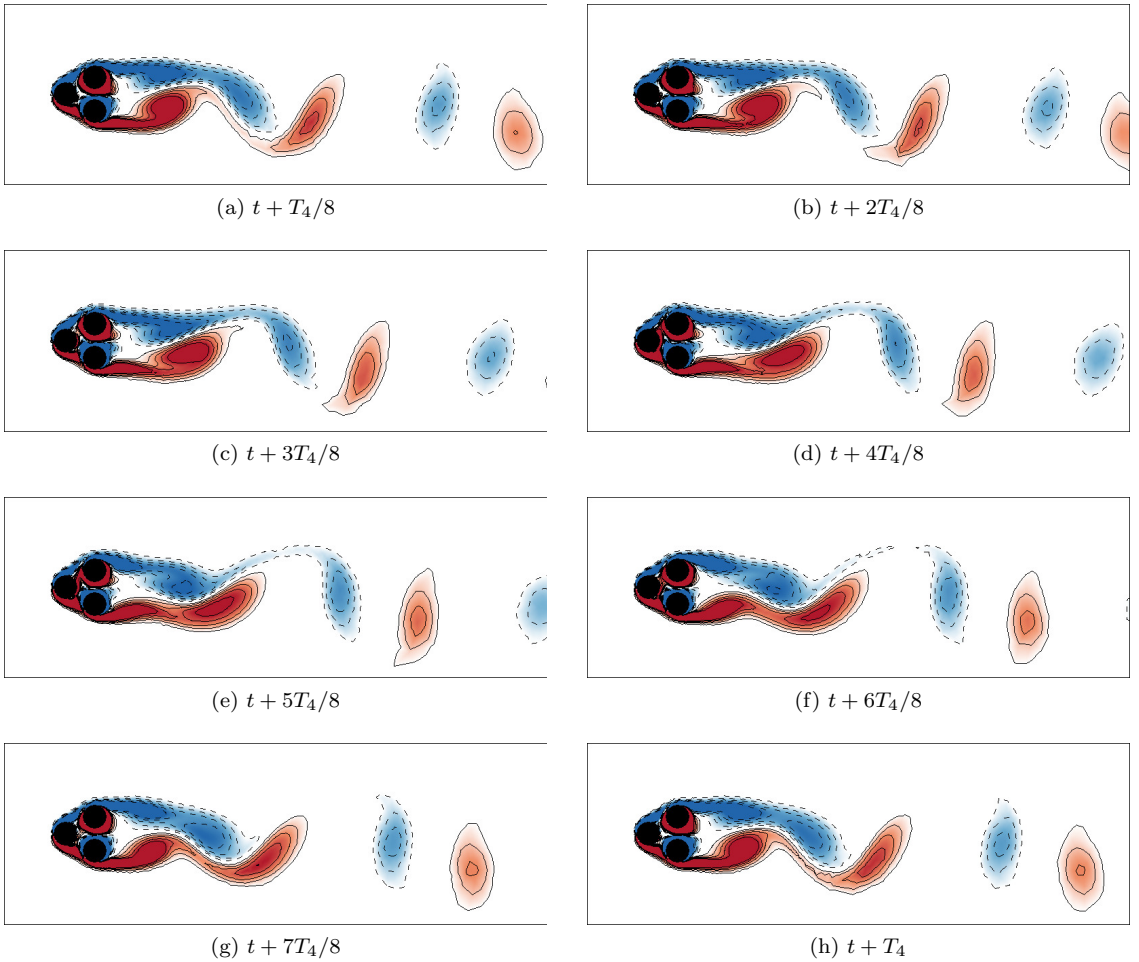


Figure 4.16: Characteristics of the flow controlled by $\boldsymbol{b}^{\mathrm{HB}}$ starting from the steady solution ($t = 0$). The transient spans until $t \approx 400$. (a) Time evolution of the lift coefficient C_L, (b) phase portrait, (c) time evolution of the drag power J_a (blue), actuation power J_b (green) and net drag power J (red) and (d) Power Spectral Density (PSD) showing the frequency $f_4 = 0.108$ and its first harmonic.



(a) $t + T_4/8$ (b) $t + 2T_4/8$

(c) $t + 3T_4/8$ (d) $t + 4T_4/8$

(e) $t + 5T_4/8$ (f) $t + 6T_4/8$

(g) $t + 7T_4/8$ (h) $t + T_4$

Figure 4.17: Vorticity fields of the flow controlled by b^{HB}. (a)-(f) Time evolution of the vorticity field in the last period of the 1000 time units simulation. The color code is the same as figure 4.1. T_4 is the period associated to the frequency f_4.

4.6 Summary

Search space	Control law	J/J_0	J_a/J_0	J_b/J_0
Unforced natural	-	1	1	0
Symmetric steady	$b_1 = 0$, $b_2 = -b_3 = \text{cst}$	0.7652	0.5695	**0.1956**
Multi-frequency forcing	$\boldsymbol{b}(t) = K(\boldsymbol{h}(t))$	0.7476	0.5109	0.2368
Feedback control	$\boldsymbol{b}(t) = K(\boldsymbol{s}(t))$	0.7451	**0.4744**	0.2706
General control	$\boldsymbol{b}(t) = K(\boldsymbol{s}(t), \boldsymbol{h}(t))$	**0.7363**	0.4845	0.2518

Table 4.5: Summary of the performances for the best solutions of each type of optimization. The bold values are the best for each cost.

In this chapter, we apply the MLC methodology previously described to minimize the net drag power of the fluidic pinball. First, a parametric study on the subspace of symmetric steady forcing supports that the boat tailing configuration appears as a key strategy to reduce the drag power. Then three search spaces are explored, first we allow for multi-frequency forcing, then we optimize a feedback control law, and finally, we allow both strategies for a hybrid optimization. All three optimizations built control laws that include a boat tailing-like structure and discard the open-loop periodic functions when they are in the function library. However, we notice that an asymmetry in the boat tailing is systematically favored. Some improvement can be achieved with the addition of sensor information, reducing the cost from $J_{\text{MF}}/J_0 = 0.7476$ to $J_{\text{HB}}/J_0 = 0.7363$. The costs of all runs are summarized in table 4.5. Thus, in less than 1000 evaluations, MLC managed to build a control combining asymmetric boat tailing and phasor control to reduce the net drag power in a model-free and with very few knowledge a priori. MLC rediscovers, in particular, that to delay the vortex shedding, one can re-energize the shear layer with periodic forcing and that vectoring the flow towards the centerline helps to increase the base pressure. The hybrid control built with MLC achieves the most net drag reduction so far.

In this part, we unveiled the forces at play in the learning process of genetic programming and applied to the reduction of the net drag power of the fluidic pinball. The parametric study of MLC, carried out in App. A.2, revealed the importance of key meta-parameters. Such analysis serves as a guide to select adequate parameters for future MLC studies. Applied to the fluidic pinball, MLC successfully managed to build control laws in different search spaces in less than 1000 evaluations, revealing key actuation mechanisms, without any prior knowledge, and combining them for further performance: the best control comprises a combination of asymmetric boat tailing and phasor control.

Chapter 5

Conclusions and outlook

We have introduced Machine Learning Control (MLC) based on linear genetic programming for a simple dynamical system. This example is easily reproducible with the provided and documented Matlab code. This code or small variations thereof have been applied in dozens of plants: nonlinear dynamical systems, direct numerical simulations and turbulence control experiment (Noack, 2019). The meta parameters have hardly been changed. The algorithm has reliably and repeatability converged to an assumed global minimum of the cost function. Hence, MLC can be expected to be successfully applied in many other nonlinear dynamics systems and flow control simulations or experiments without the need of change of meta parameters, used functions, etc. While linear control theory provides unrivaled methods for linear dynamics, MLC becomes more and more competitive with increasing control complexity, e.g., by the degree of plant nonlinear, dimension of the dynamics, number of actuation commands and sensing signals and also nonlinearity of the control law.

Yet, a few words of caution are in order. First, MLC is a stochastic optimization method for a nonlinear plant and comes hence without any performance guarantees, like convergence against a global minimum. Second, the typical number of sensors and actuators was $O(1) - O(10)$. The learning time, i.e. the number of individuals to be tested before convergence, is observed to slightly increase with the number of actuators and sensors. The number of actuation commands is more critical than the number of sensors. Third, a sensitive control problem where a small change of the control law implies a large change in performance may not be best targets of vanilla MLC versions and requires caution with any other control optimization/design as well. Fourth, in engineering applications the performance of the optimized control law might be affected by a change of initial conditions, measurements noise, plant uncertainty and changing operating conditions. As rule of thumb, a control law which employs large-scale/dominant structures can be expected to be robust against such changes while actuation involving small-scale/secondary structures may be fragile.

Machine learning control is an emerging rapidly evolving field overcoming some key challenges of linear control theory. Research needs to be done on many fronts, inspired by the success stories of linear control:

1. *Control landscape:* Visualization of the learning process. A starting point is offered in the first book (Duriez et al., 2017).

2. *Human interpretable control laws.* A cluster-based visualization (Cornejo Maceda et al., 2021; Castellanos et al., 2022) appears always doable and insightful.

3. *Significant reduction of testing time.* Here, myriad ideas come to mind, e.g., (1) surrogate modeling of the cost function as function of the control, (2) model identification from the MLC run as surrogate test plant, (3) pretesting of the control law to remove the time consuming testing of similar or unpromising laws.

4. *Robustness against measurement noise, plant uncertainty and changes of operating conditions.* Currently, learning the control laws under varying operating conditions and with measurement noise are a viable tested options.

5. *Online adaptivity inspired by extremum seeking.* The learning process for new operating conditions ideally requires only small changes in the operating plant. Gradient-enriched MLC (Cornejo Maceda et al., 2021) offers one avenue with subplex optimization on a chosen subspace of control laws.

6. *Transfer learning.* How can the control learning of a new operating conditions be shortened knowing the MLC run of another operating condition? The task poses a big challenge which seems not currently addressed by any approach.

7. *Failure safety of actuators and sensors.* In praxi, actuators and sensors may fail, requiring a prompt alternative control logic with the new reduced hardware. This is also one of the big challenge problems formulated already in first book on genetic programming control (Dracopoulos, 1997).

An increasing community of researchers are working on all these research tasks. Enjoy working with the MATLAB program `xMLC` and stay tuned for the progress to come!

Acknowledgements

This work is supported by the National Science Foundation of China (NSFC) through grants 12172109 and 12172111, by the Natural Science and Engineering grant 2022A1515011492 of Guangdong province, P.R. China.

Appendix A

MLC parametric dependency

In this appendix, we report the influence of some key parameters of MLC on the optimization process. In particular, we study the influence of the genetic operator probabilities ($P_{\text{Cros}}, P_{\text{Mut}}, P_{\text{Rep}}$) (Sec. A.2) and the balance between population size and number of instructions (Sec. A.3). The parametric study is carried on the stabilization of the damped Landau oscillator for its short evaluation time and its relatively simple dynamics. The goal of this section is to establish some rules of thumb for the selection of these parameters. First, we describe the optimal linear solution (Sec. A.1), employed as reference for the future optimized control laws.

A.1 Control benchmark and optimal linear solution

A.1.1 Control benchmark

The control benchmark for this parametric study is the stabilization of the damped Landau oscillator described in Sec. 3.3.1. For this study, we employ the same cost function except that we choose the penalization parameter γ equal to 1 to assure a fast convergence towards the fixed point. Thus the cost function employed is:

$$\begin{aligned} J &= J_a + J_b \\ J_a &= \frac{1}{T_{max}} \int_0^{T_{max}} (a_1^2 + a_2^2) d\tau \\ J_b &= \frac{1}{T_{max}} \int_0^{T_{max}} b^2 d\tau \end{aligned} \tag{A.1}$$

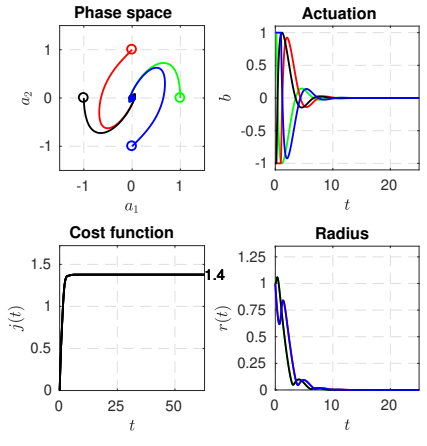
A.1.2 Optimal linear solution

We compute the optimal linear control of the control problem thanks to the `fminsearch` function of MATLAB. `fminsearch` is a derivative-free method based on Nelder-Mead simplex method (Nelder and Mead, 1965). We optimize the parameters of a linear control to stabilize the Landau oscillator. The optimal linear control is detailed in equation (A.2). This solution has been found with a random initialization of the `fminsearch` function. This local minimum may not be the global minimum of the minimization problem but for the following we will refer to it as the 'optimal linear control'. In fact, the control law learned in the following sections, reveal that this solution is not the optimal linear control as another linear solution performs better. For now, we only focus on the solution in equation (A.2), that is only a local minimum. The associated cost is $J_{\text{opt}} = 3.3403$ which corresponding to a $\Delta J_{\text{opt}}/J_0 = 94.70\%$ reduction of the unforced cost.

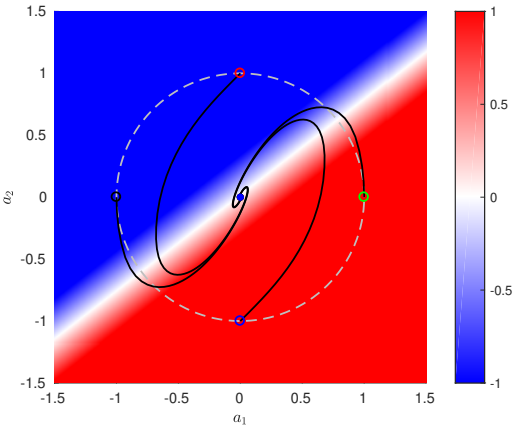
$$b_{\text{opt}} = 2.4061 a_1 - 3.0984 a_2 \tag{A.2}$$

Figure A.1 shows the controlled Landau oscillator and the actuation map for the control realized with equation (A.2).

We notice that the symmetry of the problem is respected and that the system is successfully brought to the fixed point $(a_1, a_2) = (0, 0)$ in less than two periods. We recall that the actuation effect is to push the system upwards ($b > 0$) or downwards ($b < 0$). The ratio of the coefficients in front of a_1 and a_2 define the angle for the separatrix dividing the space in two regions of positive and negative actuation. The magnitude of the coefficients define the increase rate of the actuation level as the system moves away from the separatrix. It is both the optimization of the separatrix angle and the actuation increase rate



(a) Phase space, actuation command, instantaneous cost function and radius.



(b) Visualization of the control law in the phase space. The limit cycle is depicted with a dashed line.

Figure A.1: Figures characterizing the optimal linear solution.

that brings the system faster towards the fixed point. It is worth noting that the optimal control leads the system beyond the limit cycle for a short period of time. Evidently, if the control would be on the first equation instead of the second, the results would be symmetrical to the $a_1 = a_2$ axis.

Our goal is then to stabilize the Landau oscillator with MLC and to find a better solution than the optimal linear control.

A.2 Influence of genetic operators

In this section, we stabilize the damped Landau oscillator with increasingly complex MLC algorithms. The starting point is the special case of MLC, where only one generation is employed. This case corresponds to a Monte Carlo sampling. Then, we progressively include the genetic operators (crossover, mutation and replication) and analyze their influence on the algorithm's performance.

A.2.1 Monte Carlo sampling

As detailed in Sec. 2.2, a Monte Carlo sampling consists in generating random matrices for each individual. Thus, we must first define the search space, i.e., the space of candidate solutions to the problem.

The search space

Parameter	Description	Value
	Function library	$F_1 = \{+, -, \times, \div\}$
$\boldsymbol{s}$	Controller inputs	a_1, a_2
N_{Var}	Number of variable registers	3
N_{Cst}	Number of constant registers	3
$N_{\text{Instr,max}}$	Max. number of instructions	5

Table A.1: Parameters for the control ansatz.

In the linear genetic programming framework, the search space is defined by the function and inputs library. The other parameters that can influence the exploration of the landscape are the number of variable registers N_{Var}, the number of constant registers N_{Cst} and the maximum number of instructions $N_{\text{Instr,max}}$. Indeed if N_{Var}, N_{Cst} and $N_{\text{Instr,max}}$ are small then the control laws are bound to contain only few operations, thus limiting the complexity of the accessible control laws. This aspect is further studied in App. A.3. Table A.1 summarizes the parameters chosen for the Monte Carlo sampling.

Based on these parameters, we can compute the total number of control laws in the search space by listing all the possible combinations. For this, we multiple the possible values for each columns of the instruction matrix and elevate the result at the power of the number of rows in the matrix. As we allow matrices of different sizes, we need to add the results for each possible number of rows. Expression (A.3) gives the order of magnitude of the number of possible control laws with the chosen parameters.

$$\sum_{q=1}^{N_{\text{Instr,max}}} [N_{\text{R}} \times N_{\text{R}} \times N_{\text{O}} \times N_{\text{Var}}]^q \approx 1.5 \times 10^{13} \tag{A.3}$$

where

$N_{\text{Instr,max}}$: maximum number of instructions.
N_{R}: total number of registers: $N_{\text{R}} = N_{\text{Var}} + N_{\text{Cst}}$.
N_{O}: number of functions in the library or mathematical operators.
N_{Var}: number of variable registers, where to store intermediate calculations.

This number 1.5×10^{13} represents the total number of matrices, i.e., the total number of accessible control laws. For this estimation, we assume that all registers are initialized with different values, which is often the case in practice, except for the output registers, i.e., the registers that store the control laws, that are initialed with zeros. Table A.2 presents the initialization of the registers for the oscillator

Variable registers		Constant registers	
r_1	0	r_4	$c_1 = -0.94$
r_2	s_1	r_5	$c_2 = -0.05$
r_3	s_2	r_6	$c_3 = -0.72$

Table A.2: Initialization of the registers. The constants c_i have been randomly taken in the range $[-1, 1]$.

stabilization problem before the execution of the instruction matrices.

Among the 1.5×10^{13} possible control law, there are, of course, a lot of identical and equivalent control laws, due to overwriting, useless instructions, multiplication by 0, etc. This constitutes a valuable feature as the expression of the global minimum is no longer unique and can be built in different ways, thus reducing the 'size' of the search space.

A randomly generated control law

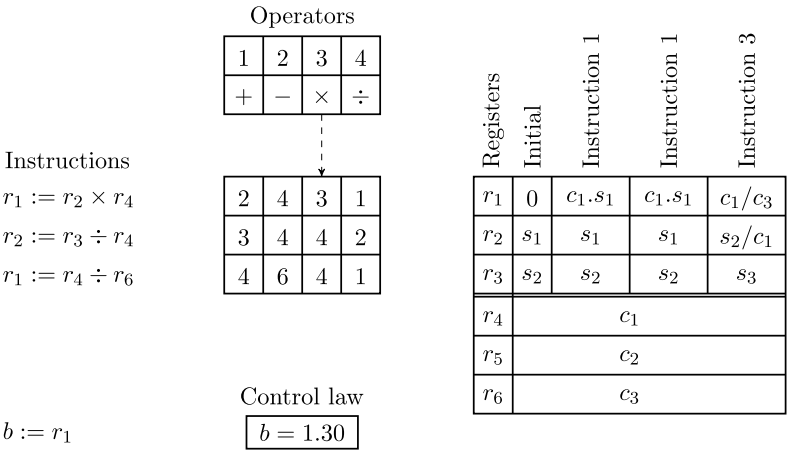
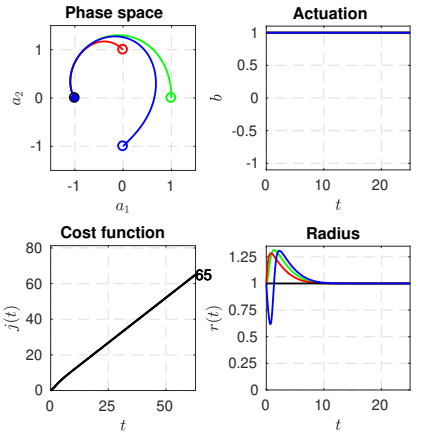
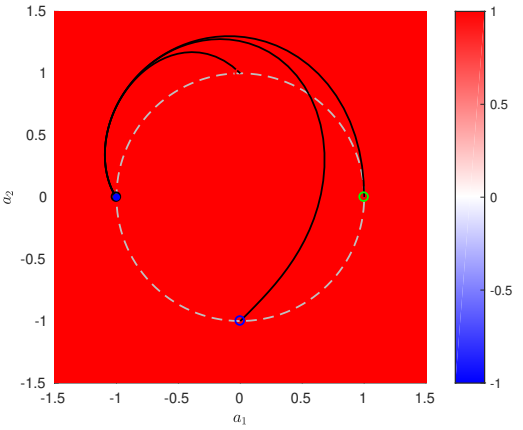


Figure A.2: Example of randomly generated matrix and its translation to a control law.

Figure A.2 shows an instruction matrix randomly generated and depicts the process of translation. The resulting control is a constant beyond the actuation limits, thus this control law is equivalent to a constant forcing at $b = 1$. Before performing a Monte Carlo sampling, let's first generate a random control law. Figure A.3 gives a visualization of the control law. To avoid controls with too strong actuations, we limit the actuations between -1 and 1 with a clip function. We notice that with such action, the point



(a) Phase space, actuation command, instantaneous cost function and radius.



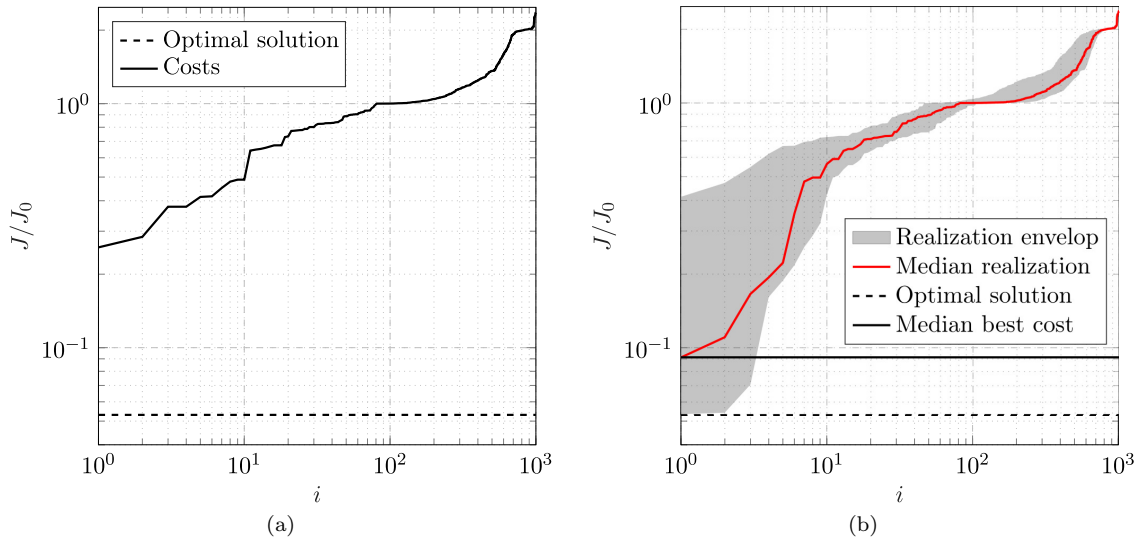
(b) Visualization of the control law in the phase space. The limit cycle is depicted with a dashed line.

Figure A.3: Visualization of the randomly generated control law $b = 1.30$. The control being limited between -1 and 1, this control is equivalent to 1.

$(a_1, a_2) = (-1, 0)$ becomes a fixed point. Indeed, without control this point is the unique point where $(\dot{a}_1, \dot{a}_2) = (0, -1)$, thus when adding a constant forcing $b = 1$, this point becomes a fixed point.

Monte Carlo sampling

Now, let's perform a Monte Carlo sampling with the parameters of A.1. In order to emulate experimental conditions, we only perform $N_i = 1000$ evaluations of the cost functions. Thus, we only generate 1000 control laws to be evaluated. Our Monte Carlo implementation is optimized with the screening of redundant individuals. More informations are provided in App. D. Figure A.4a shows the distribution of



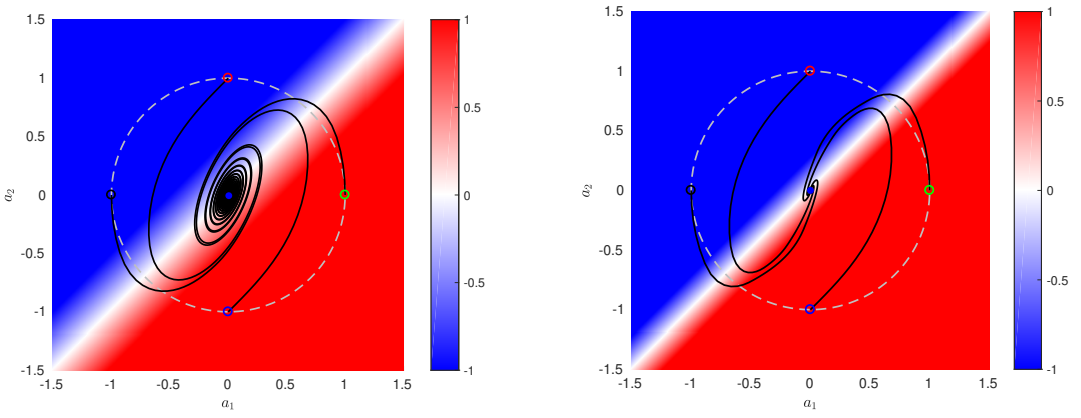
(a) (b)

Figure A.4: (a) Distribution of the cost of 1000 randomly generated control laws for the stabilization of the damped Landau oscillator. (b) Envelop of 100 realizations of Monte Carlo sampling for the same problem. The median realization ($\rho = 50^{\text{th}}$ best) is depicted in red. The black horizontal line represents the cost of the best individual of the median realization. The associated cost reduction is $\Delta J/J_0 = 91.17\%$. The dashed line represents the cost of the optimal linear solution ($\Delta J_{\text{opt}}/J_0 = 94.70\%$). The vertical and horizontal axis are in log scale. In both (a) and (b) the individuals are sorted following their cost.

costs in log-log scale for one realization of Monte Carlo sampling. The values are sorted and normalized with the unforced cost J_0. We notice in figure A.4a, that the performance of the best solution after 1000

evaluated control laws is not as good as the optimal linear solution. The associated cost reduction is $\Delta J/J_0 = 74\%$.

However, as we rely on a stochastic process, only one realization of the optimization process is not enough to estimate the performance of the method. That is why, we perform $N_\rho = 100$ realizations of Monte Carlo sampling to have statistically relevant realizations. Figure A.4b shows the envelop of the one hundred realizations of Monte Carlo. The cost reduction of the best individuals in the realization envelop goes from 58.55% to 94.64%, and the median performance reduces the cost by $\Delta J/J_0 = 91.17\%$. In the following the realizations are indexed by ρ and the median performance is defined as the $\rho = 50^{\text{th}}$ best realization. The best control law of the median Monte Carlo realization is



(a) (b)

Figure A.5: Visualization of control laws in the phase space. (a) Best control law of the median Monte Carlo realization (A.2.1). (b) Best control law of the best Monte Carlo realization (A.2.1). The limit cycle is depicted with a dashed line.

$$b_{\text{MC,median}} = 2.0907(a_1 - a_2)$$

and the best control law of the best Monte Carlo realization is

$$b_{\text{MC,best}} = 3.7747(a_1 - a_2);$$

Their associated cost reduction are $\Delta J/J_0 = 91.17\%$ and $\Delta J/J_0 = 94.64\%$ respectively. The expressions have been simplified compared to the expressions computed by `xMLC`. The two controls have the same form but with a different coefficient. The coefficients are related to the width of the control band and thus at the speed of convergence of the oscillator towards the fixed point. This difference can be appreciated in figures A.5a and A.5b Figure A.6 displays the best individuals of the hundred realizations. We notice that the first twenty-three control laws are all similar. They are similar to the optimal linear control with different band width. Starting from the 33^{rd} run, the best individuals takes more than two periods to reach the fixed point. We remark a majority of linear-like solutions, especially among the best ones. The first nonlinear solution is ranked 49^{th} and the other ones are ranked beyond the 80^{th} realizations. We also notice that those nonlinear solutions still present a symmetry in their actuation map. Among the least performing realizations, there are 15 control laws that are equivalent to $-a_2$ to within one multiplicative positive constant, resulting on horizontally symmetric solutions. We can expect that the genetic operators in the following sections combines appropriately a_1 and a_2 for as faster stabilization.

The study shows that a Monte Carlo sampling is able to build a good performing control laws for such problem. However, only 1/3 of all realizations achieved a stabilization in less than 2 periods. We note in particular that the best control law of the median realization is far from an efficient solution as the solution takes many periods to reach the fixed point. Monte Carlo sampling lacks reproducibility on the ability to build efficient solutions. In the next sections, crossover and mutation are included progressively in the learning process, showing that they allow much better performances in terms of speed, solution and reproducibility.

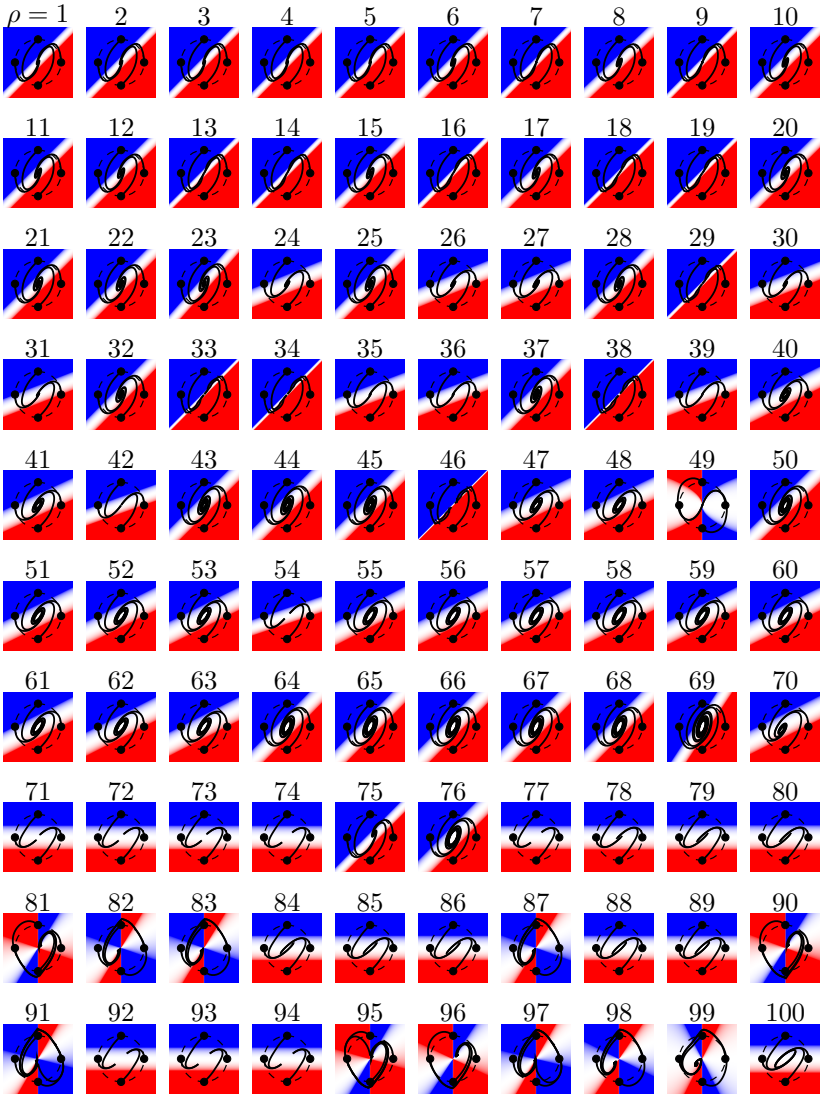


Figure A.6: Visualization of the best control laws for each one of the hundred 100 Monte Carlo realizations. The index ρ denotes the different realizations. The realizations are sorted by the cost of their best individual. 1 being the best realization and 100 the worst. For simplicity, the trajectories for the four initial conditions are only plotted for the two first periods only.

A.2.2 Exploitation with crossover

In this section, we improve the Monte Carlo sampling by introducing the crossover genetic operator. We follow the algorithm described in figure 2.6 but only considering the crossover operator. For this, we set the genetic operators probabilities to 0 except for the crossover, set to 1. As we rely on a evolution process, we also set elitism to 1 to assure that the best individual is saved throughout the generations. The parameters related to the evolution process are detailed in A.3. The remaining parameters are to same as the one used for the Monte Carlo sampling, see table A.1. Like for the Monte Carlo sampling, $N_\rho = 100$ realizations of the crossover-only MLC are performed. Each realization consist of a population of $N_{\text{Pop.size}} = 100$ individuals evolving through $N_{\text{G}} = 10$ generations for a total of $N_i = 1000$ evaluations. Figure A.7 summarizes the parameters for this study.

Like for Monte Carlo sampling, the 100 realizations are collapsed in a single figure A.8. First, we note that the median realization of crossover-only MLC performs better than the Monte Carlo sampling one. Second, the convergence speed is improved, indeed already after 700 evaluations, the cost of the best individual is lower compared to the Monte Carlo sampling. This shows that a genetic programming algorithm only including recombination with crossover is already able to stabilize the Landau oscillator and performs better than Monte Carlo sampling.

Parameter	Description	Value
$N_{\text{Pop.size}}$	Population size	100
N_{G}	Number of generations	10
N_{Tour}	Tournament size	7
N_{E}	Elitism	1
P_{Cros}	**Crossover probability**	1
P_{Mut}	Mutation probability	0
P_{Rep}	Replication probability	0

Table A.3: Parameters for MLC with crossover only.

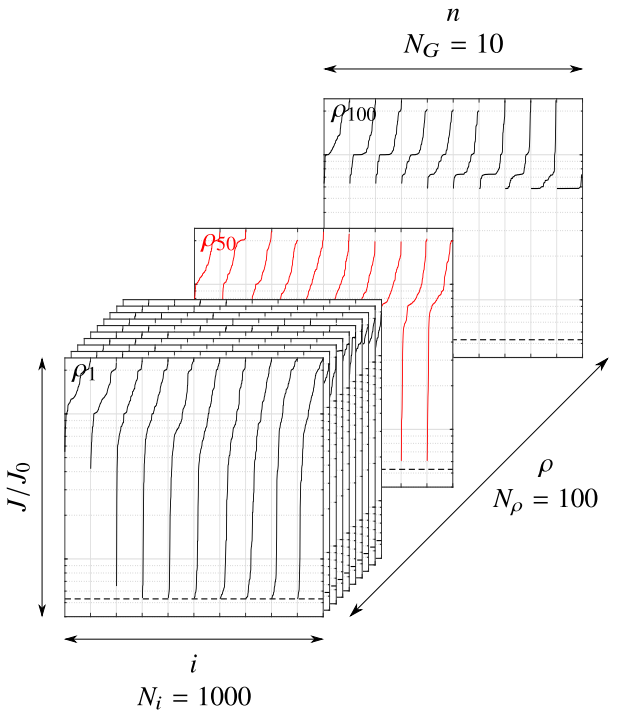


Figure A.7: Conceptual figure of the $N_\rho = 100$ realizations of MLC. Each graph represents a realization including $N_{\text{Pop.size}} = 10$ generations of 100 individuals. The median realization is denoted $\rho = 50$ and colored in red.

The resulting algorithm is an exploitative-only or crossover-only MLC. To have a fair comparison between Monte Carlo and MLC, we evaluate $N_i = 1000$ individuals distributed in a population of $N_{\text{Pop.size}} = 100$ individuals that evolves 9 times for a total of $N_\rho = 10$ generations.

The best control law of the median crossover-only MLC is

$$b_{\text{C,median}} = a_1 - 2.4876a_2 \tag{A.4}$$

and the best control law of the best realization is

$$b_{\text{C,best}} = 0.63361(4a_1 - 5a_2); \tag{A.5}$$

Their associated cost reduction are $\Delta J/J_0 = 93.94\%$ and $\Delta J/J_0 = 94.69\%$ respectively. After simplification of the expressions, both control laws are actually linear with a cost reduction close to the optimal linear solution ($\Delta J_{\text{opt}}/J_0 = 94.70\%$). We note, in particular, in figures A.9a and A.9b that the separatrix angle is different for the two solutions leading to a faster convergence for the best realization. Moreover, figure A.10 shows that 59 of the 100 realizations optimized control laws able to stabilize the oscillator in less than 2 periods. Most of the best control laws derived are linear and close to the optimal linear one, except for realizations 24 and 54 that presents nonlinear behaviors. The note that there is around the same number of horizontally symmetric solutions (13) compared to Monte Carlo sampling. An explanation may be a poor first generation of individuals that are not able to build better control laws with crossover only. The explorative power of mutation is expected to generate new individual and more complex individuals to enrich the population.

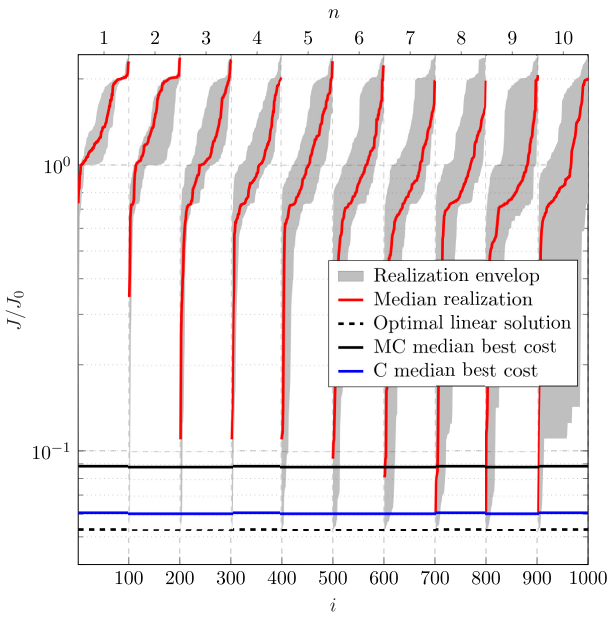


Figure A.8: Costs of individuals for 100 realizations of crossover-only MLC. The gray region represents the envelop of the 100 realizations. The individuals are divided in 10 groups of 100 individuals representing the 10 generations. In a given generation the individuals are sorted by their cost. The median realization is depicted in red. The black horizontal line represents the median cost for the Monte Carlo sampling. The blue horizontal line represents the median cost ($\Delta J/J_0 = 93.94\%$) for the crossover-only MLC. The black dotted line represents the cost of the optimal linear control law. The vertical axis is in log scale.

The exploitative-only or crossover-only MLC shows that only with one genetic operator genetic programming is more performing than Monte Carlo sampling. Crossover-only MLC builds better control laws are optimized with less evaluations and has a better reproducibility. Adding an explorative operator is expected to improve the results.

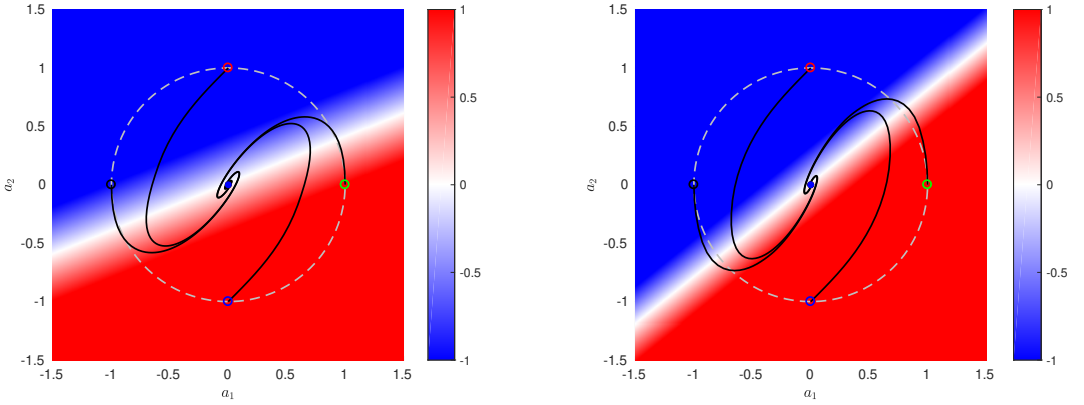
A.2.3 Exploitation and exploration with crossover and mutation

Parameter	Description	Value
$N_{\text{Pop.size}}$	Population size	100
N_{G}	Number of generations	10
N_{Tour}	Tournament size	7
N_{E}	Rlitism	1
P_{Cros}	**Crossover probability**	$0, 0.1, ..., 1$
P_{Mut}	**Mutation probability**	$1 - P_{\text{Cros}}$
P_{Rep}	Replication probability	0

Table A.4: Parameters for crossover-mutation MLC

In this section, we progressively introduce exploration in the optimization process by tuning the crossover and mutation probabilities. P_{Cros} goes from 1 to 0 by steps of 0.1 and P_{Mut} increases accordingly. Thus, for $(P_{\text{Cros}}, P_{\text{Mut}}) = (1, 0)$ MLC is purely exploitative, like in the previous section, and for $(P_{\text{Cros}}, P_{\text{Mut}}) = (0, 1)$ MLC is purely explorative. Table A.4 summarizes the parameters employed for this section.

In total 11 configurations are tested. For each couple of $(P_{\text{Cros}}, P_{\text{Mut}})$ 100 realizations have been carried out. Figure A.11a summarizes the performances of the median realization for each configuration. We recall that in this study we define the median as the 50^{th} realization. First, we note that an exploration-only MLC based only on mutation performs worse than any other combination but still better than Monte Carlo sampling. This shows that random 'mutation' of the best individuals throughout the generations is able to improve the performance of individuals and is better than just a random sampling. However, exploration-only MLC is less performing than exploitation-only MLC. This may be explained



(a) (b)

Figure A.9: Visualization of control laws in the phase space. (a) Best control law of the median crossover-only MLC realization (A.4). (b) Best control law of the best crossover-only MLC realization (A.5). The limit cycle is depicted with a dashed line.

by the fact that crossover recombines individuals that are known to be good whereas mutation still relies on randomness to improve its best individuals. Second, we note that any combination of crossover and mutation performs better than crossover and mutation only. In particular, they are 4 configurations $P_{\text{Mut}} = 0.2, 03, 04, 0.5$ that reduce the cost to the same level and close to the performance of the optimal linear control solution. The two best ones being $P_{\text{Mut}} = 0.3$ and $P_{\text{Mut}} = 0.4$. Figure A.11b shows the learning process of the $(P_{\text{Cros}}, P_{\text{Mut}}) = (0.7, 0.3)$ configuration. We note in particular that compared to crossover-only MLC, this configuration learns better control laws and with less evaluations. Indeed, already after 500 evaluations, an individual performing better than crossover-only MLC has been found.

Figure A.13b shows that for the $(P_{\text{Cros}}, P_{\text{Mut}}) = (0.7, 0.3)$ configuration, 79 of the 100 realizations optimized control laws able to stabilize the oscillator in less than 2 periods. Again, like for crossover-only MLC, most of the best control laws derived are linear and close to the optimal linear one, except for realizations 75 and 79 that presents nonlinear behaviors. Moreover, 6 realizations optimized horizontally symmetric control laws, which is half the number of crossover-only MLC and Monte Carlo sampling. This shows that the introduction of mutation improves indeed the exploration capability of MLC.

Equation (A.6) is the control law that has been found by the median realizations of the four best configurations in figure A.11a.

$$b_{\text{CM,median}} = 2a_1 - 3a_2 \tag{A.6}$$

This same control law has been in the four configuration. The coefficients have been built by additions and subtraction of a_1 and a_2 only, these realizations ignored the random constants in the registers. The best control law of the best realization of $(P_{\text{Cros}}, P_{\text{Mut}}) = (0.7, 0.3)$ MLC configuration is:

$$b_{\text{CM,best}} = 2.4932a_1 - 3.1001a_2. \tag{A.7}$$

Again, it is a linear control but whose coefficients have been build from the random constants. The cost reduction associated to the median and best realization are, respectively, $(\Delta J/J_0 = 94.62\%)$ and $(\Delta J/J_0 = 94.69\%)$. Figures A.13a and A.13b displaying their corresponding visualizations. The only difference noticeably is the angle of the separatrix.

We've shown that combining crossover and mutation allows better performances than crossover and mutation alone. Control laws performing as well as the optimal linear control are built with less evaluations. The reproductivity of the results have also been improved as there are more realizations being able to quickly stabilize the oscillator. In the next section, we analyze the full-fledge MLC, i.e., including crossover, mutation and replication.

A.2.4 MLC Optimization with crossover, mutation and replication

In this section, we study the influence of the operators probability $(P_{\text{Cros}}, P_{\text{Mut}}, P_{\text{Rep}})$. For this, we run MLC optimizations with different combinations of $(P_{\text{Cros}}, P_{\text{Mut}}, P_{\text{Rep}})$. We start from $(P_{\text{Cros}}, P_{\text{Mut}}, P_{\text{Rep}}) = (1, 0, 0)$, full crossover, and we increase and decrease each parameter with a step of 0.1. We recall that

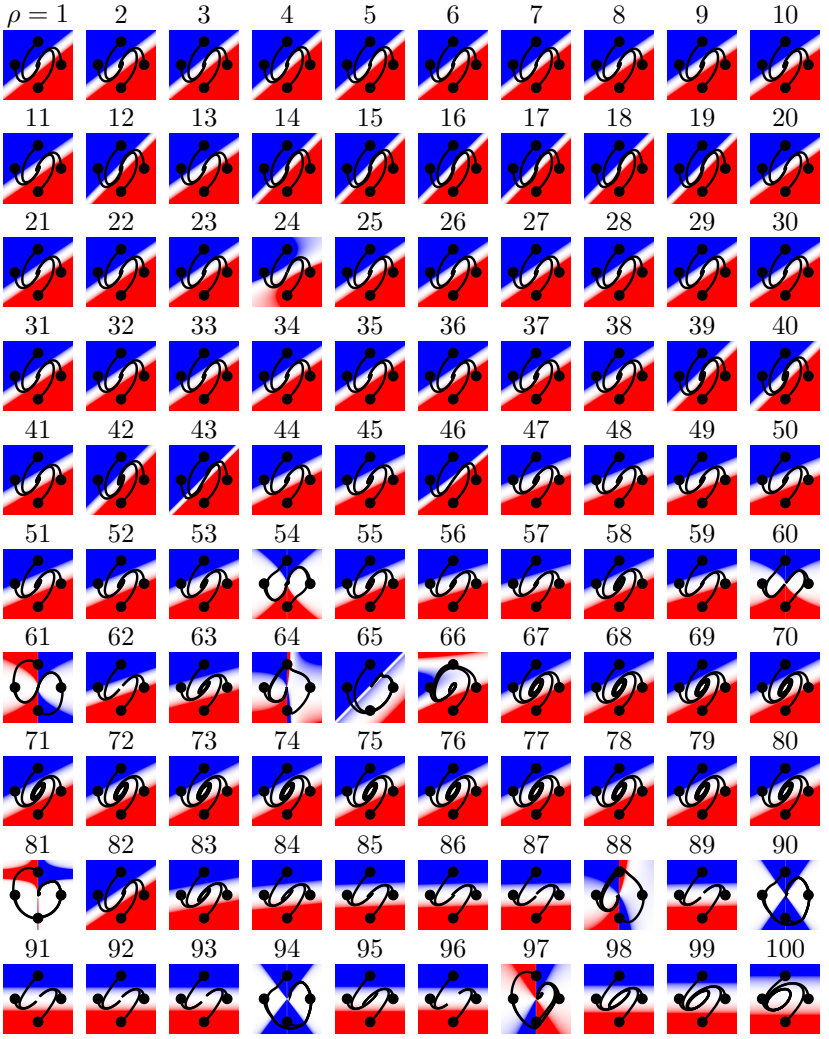


Figure A.10: Same as A.6 but for crossover-only MLC.

$P_{\text{Cros}} + P_{\text{Mut}} + P_{\text{Rep}} = 1$. There are in total 66 combination of parameters to test. We run $N_\rho = 100$ realizations for each combination of probabilities. The parameters are summarized in table A.5.

Figure A.14 presents the results of the 66 configurations. The replication-only is, as expected, the worse configuration. Indeed, this is equivalent to run a Monte Carlo sampling with only $N_i = 100$ individuals. The addition of crossover and mutation progressively improves the replication-only optimization. We notice that both crossover-only and mutation-only MLC are less efficient than most of the combinations of crossover, mutation and replication. Also, it is the combination of crossover and mutation that gives the best results. We note that from configuration $(P_{\text{Cros}}, P_{\text{Mut}}, P_{\text{Rep}}) = (0.4, 0.1, 0.5)$, ranked 52^{nd}, the median costs are lower than the median cost of Monte Carlo sampling.

It is worth noting that crossover-only MLC (ranked 43^{rd}) is better than mutation-only MLC (ranked 47^{th}). This indicates that crossover and exploration has a better learning potential than the exploitative power of mutation.

Also, we highlight that the median cost of the 21 first configurations are all equal. Indeed, the median realizations of these 21 configurations all manage to build the same controller, in 13 different ways:

$$b = 2a_1 - 3a_2$$

that reduces the cost by $\Delta J/J_0 = 94.62\%$, see Sec. A.2.3 and figure A.13a. Surprisingly, even for such simple configuration, the replication operator also plays a non-negligible part as the 21 first configurations are all equivalent. We can conclude that memory is beneficial to explore a search space even when it is expected to have few minima.

To differentiate the configurations, we look at the cost of the best individual for their 25^{th} realization. The five best configurations have similar costs and are in the same region: $(P_{\text{Cros}}, P_{\text{Mut}}, P_{\text{Rep}}) =$

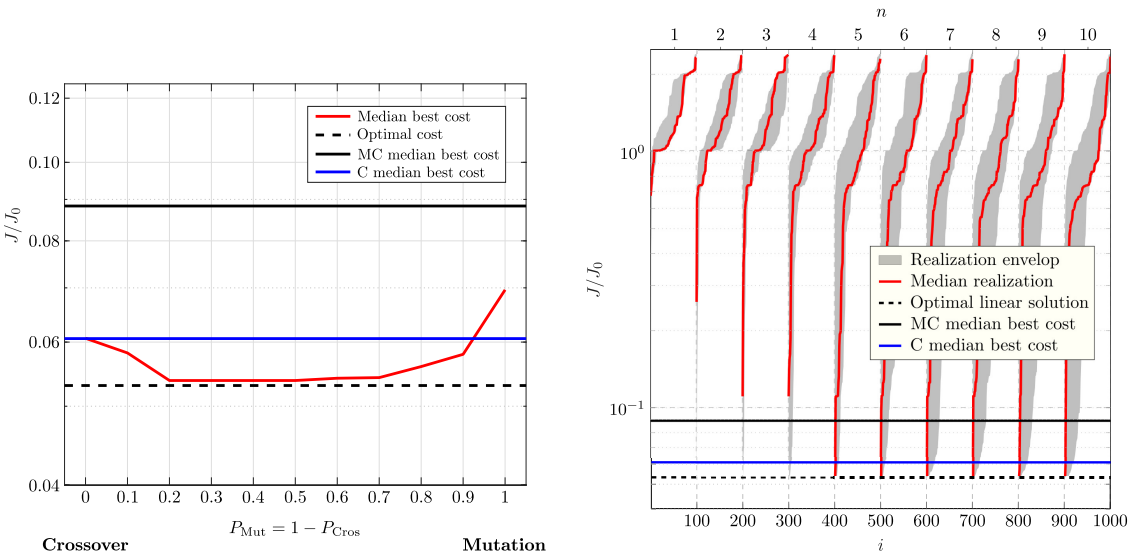


Figure A.11: (a) Performance of MLC regarding the parameters $P_{\text{Mut}} = 1 - P_{\text{Cros}}$. The most cost reduction ($\Delta J/J_0 = 94.62\%$) is achieved for configurations $P_{\text{Mut}} = 0.2$ to $P_{\text{Mut}} = 0.5$. The vertical axis is in log-scale. (b) Same figure as A.8 but for the $(P_{\text{Cros}}, P_{\text{Mut}}) = (0.7, 0.3)$ configuration of MLC. The best median best cost for crossover-mutation MLC is not displayed as it value would overlap with the cost of the optimal linear solution.

$(0.6, 0.2, 0.2)$, $(0.7, 0.2, 0.1)$, $(0.6, 0.3, 0.1)$, $(0.5, 0.2, 0.3)$, $(0.5, 0.4, 0.1)$. We choose to study the configuration at the 'center' of them all $(P_{\text{Cros}}, P_{\text{Mut}}, P_{\text{Rep}}) = (0.6, 0.3, 0.1)$. Figure A.15 shows the learning process of this configuration for the median realization. Like for crossover-mutation $(P_{\text{Cros}}, P_{\text{Mut}}, P_{\text{Rep}}) = (0.7, 0.3, 0)$ MLC, the learning is accelerated compared to crossover only and the best solution reaches reduces the cost similarly to the optimal linear solution.

Figure A.16 shows the best control laws of all realizations of $(P_{\text{Cros}}, P_{\text{Mut}}, P_{\text{Rep}}) = (0.6, 0.3, 0.1)$ MLC. We note, first that 78 realizations manage to stabilize the fixed point in less than two periods. This number is one less than the crossover-mutation $(P_{\text{Cros}}, P_{\text{Mut}}, P_{\text{Rep}}) = (0.7, 0.3, 0)$ MLC. This is not surprising as the control laws are less recombined due to the 0.1 replication probability. However, we note that there are only 2 horizontally symmetric solution, suggesting that memory benefits in building complex control laws.

The best control law all probability combinations and realizations combined is found for the config-

Parameter	Description	Value
	Function library	$F_1 = \{+, -, \times, \div\}$
$\boldsymbol{s}$	Controller inputs	a_1, a_2
N_{Var}	Number of variable registers	3
N_{Cst}	Number of constant registers	3
$N_{\text{Instr,max}}$	Max. number of instructions	5
γ	Penalization parameter	1
$N_{\text{Pop.size}}$	Population size	100
N_{G}	Number of generations	10
N_{Tour}	Tournament size	7
N_{E}	Elitism	1
P_{Cros}	**Crossover probability**	P_{Cros}
P_{Mut}	**Mutation probability**	P_{Mut}
P_{Rep}	**Replication probability**	$1 - P_{\text{Cros}} - P_{\text{Mut}}$

Table A.5: `xMLC` for the optimization. The operators probability $(P_{\text{Cros}}, P_{\text{Mut}}, P_{\text{Rep}})$ are studied in this section.

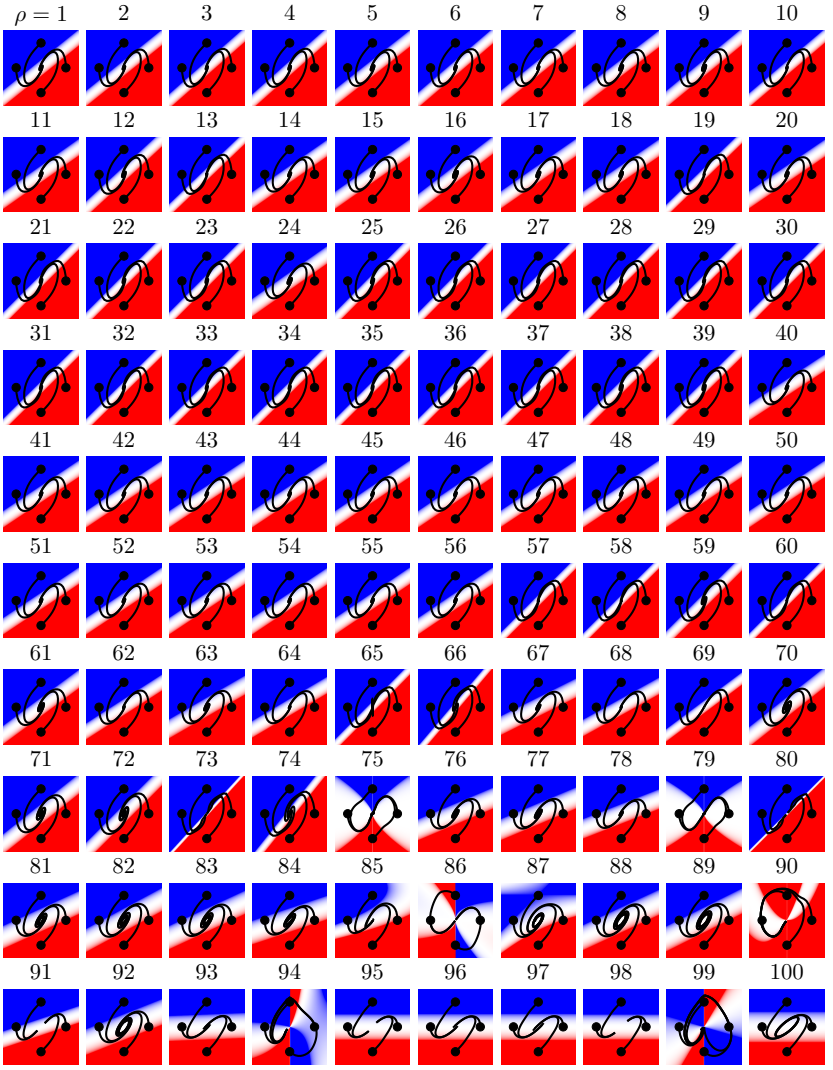


Figure A.12: Same as A.10 but for the $(P_{\text{Cros}}, P_{\text{Mut}}) = (0.7, 0.3)$ MLC configuration.

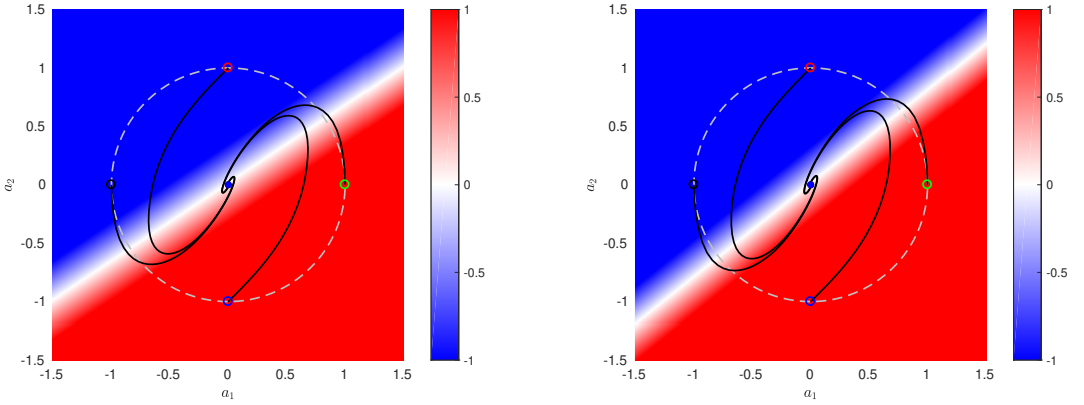
uration $(P_{\text{Cros}}, P_{\text{Mut}}, P_{\text{Rep}}) = (0.5, 0.4, 0.1)$ and reads after simplification:

$$b_{\text{best}} = \left(\frac{a_1 - 2a_2}{a_1}\right)^2 (a_1 - a_2).$$

It allows a cost reduction of $\Delta J_{best}/J_0 = 94.78\%$, better than the optimal linear control, revealing that the linear control is in fact not the global minimum of the problem. It is worth noting that b_{best} is built only from a_1 and a_2. No constants have been used to 'adjust' the control law. This may be explained by the fact that tuning constants from the initial random ones is costly in terms of instructions. Figure A.17 shows the phase portrait of the controlled oscillator with the best control found b_{best}. The actuation map displays the nonlinear effect of the control. The actuation is globally similar to the linear control, however the control intensifies near the fixed point. An absence of control in the diagonal direction prevents the system to go beyond the limit cycle unlike the optimal linear control and thus converges even faster. The curvature of the lobes that delimit the control must play a role in the fast convergence of the system towards the fixed point. We note that the oscillator converges linearly when close enough to the fixed point. Thus, `xMLC` manages to find an unexpected structure for the control law that performs better than the optimal linear control. We can expect that this nonlinear control law can be improved with adequate constant tuning.

If we take a look at the best individual of the $(P_{\text{Cros}}, P_{\text{Mut}}, P_{\text{Rep}}) = (0.6, 0.3, 0.1)$ combination, all realizations combined, we notice that the best control law is not only linear:

$$b_{(0.6,0.3,0.1),\text{best}} = 2.4353a_1 - 3.1238a_2$$



(a) (b)

Figure A.13: Visualization of control laws in the phase space. (a) Best control law of the median crossover-mutation MLC realization (A.6). (b) Best control law of the best crossover-mutation MLC realization (A.7). The limit cycle is depicted with a dashed line.

but the associated cost reduction $\Delta J/J_0 = 94.70\%$ is similar to the 'optimal' linear control found with `fminsearch`. This shows that the solution (equation (A.2)) is, in fact only a local minimum of the space of linear control laws. When we run a `fminsearch` with this new control law as initial condition, we do not observe any improvement thus we can assume that this new solution is another local minimum in the space of linear control laws. One explanation of the presence of several minima in such simple dynamical system may be the nonlinear saturation of the actuation command. We do not display the phase portrait of the best linear control found with the $(P_{\text{Cros}}, P_{\text{Mut}}, P_{\text{Rep}}) = (0.6, 0.3, 0.1)$ combination as it is identical to figure A.1 and no differences can be seen with the naked eye.

Here, we shall stop the analysis of the influence of the probability operators $(P_{\text{Cros}}, P_{\text{Mut}}, P_{\text{Rep}})$ to focus on other meta-parameters. In the following, we employ MLC with the $(P_{\text{Cros}}, P_{\text{Mut}}, P_{\text{Rep}}) = (0.6, 0.3, 0.1)$ configuration as it is one that have the lowest median cost value.

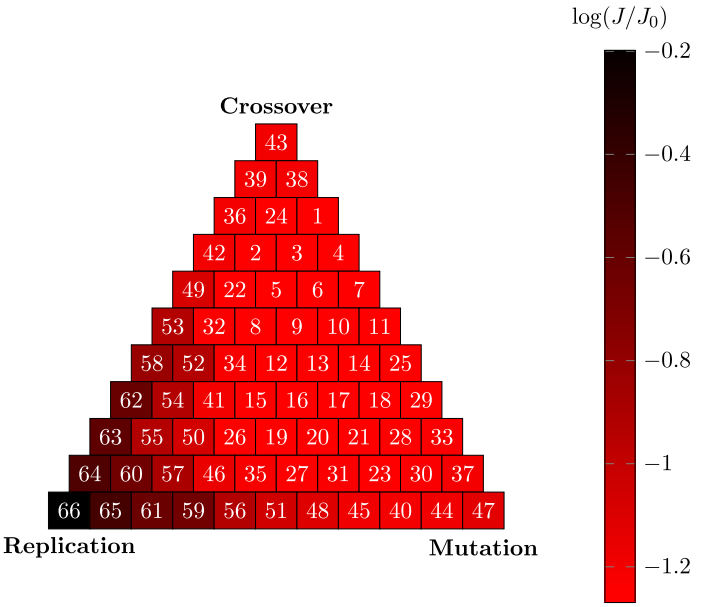


Figure A.14: Performance of MLC regarding the parameters $(P_{\text{Cros}}, P_{\text{Mut}}, P_{\text{Rep}} = 1 - P_{\text{Cros}} - P_{\text{Mut}})$. Each block of the triangle corresponds to one probability configuration. A neighbor block means a variation of 0.1 of the genetic operator probabilities. Top block is crossover-only $(1, 0, 0)$, bottom-left block is replication-only $(0, 0, 1)$ and bottom-right block is mutation-only $(0, 1, 0)$. For each configuration 100 realizations have been carried out. The color code symbolize the performance of the best individual for the median realization of each configuration. The configurations are ranked following their relative performance, 1 being the best configuration and 66 the worst. The first 21 configurations have all the same minimum value corresponding to a cost reduction of $\Delta J/J_0 = 94.62\%$. Configurations 52 to the 66 perform worse than Monte Carlo sampling.

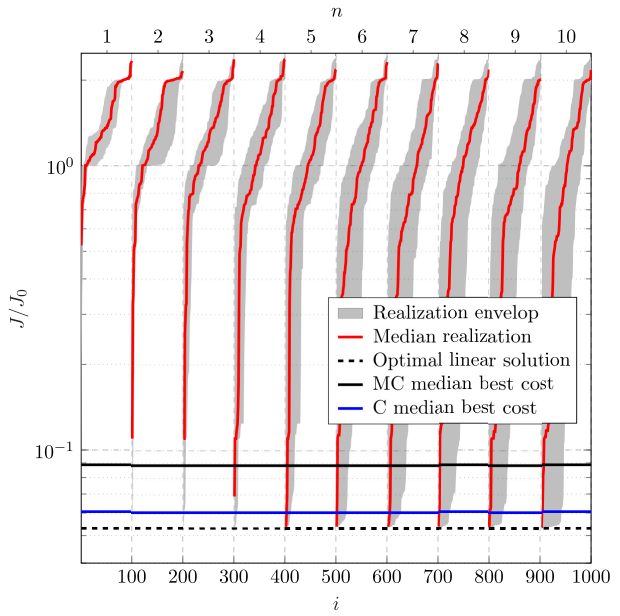


Figure A.15: Same figure as A.8 but for full-fledge MLC with the configuration $(P_{\text{Cros}}, P_{\text{Mut}}, P_{\text{Rep}}) = (0.6, 0.3, 0.1)$.

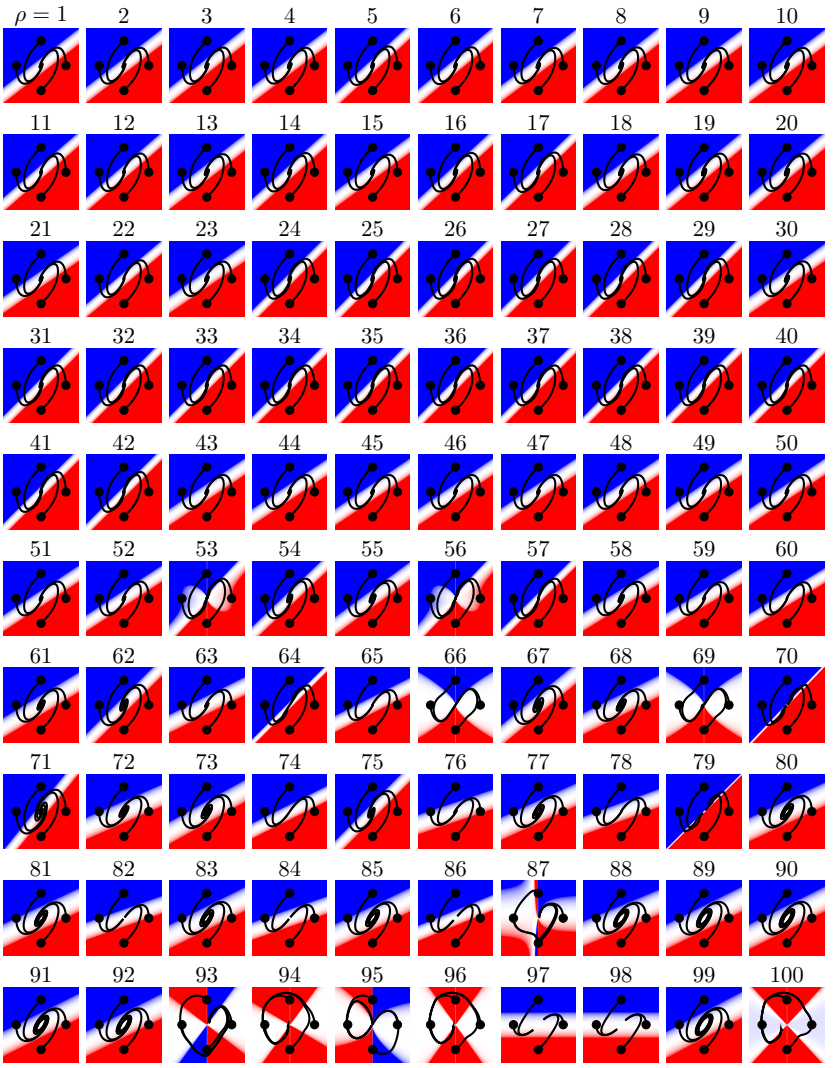
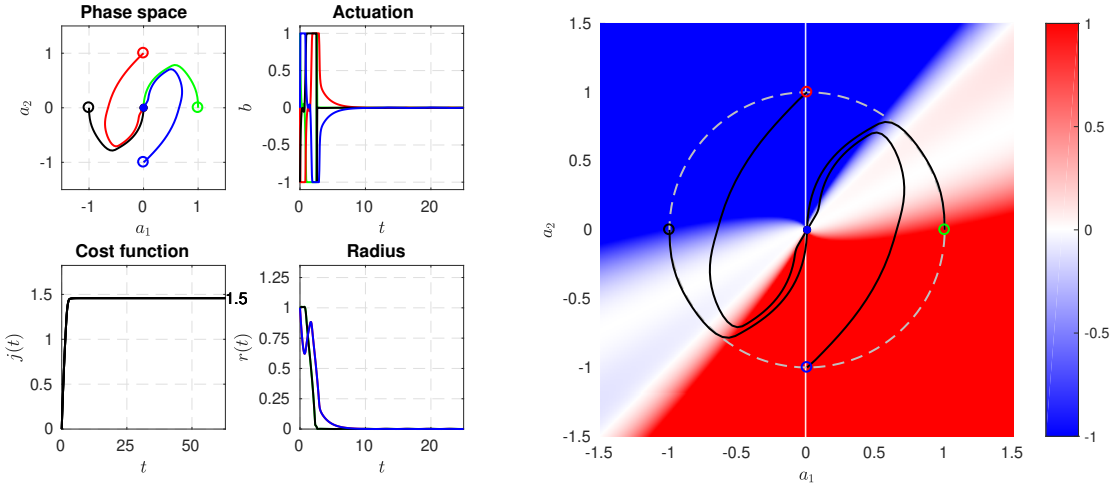


Figure A.16: Same as A.12 but for full-fledged MLC.



(a) Phase space, actuation command, instantaneous cost function and radius for the controlled case.

(b) Visualization of the control law in the phase space.

Figure A.17: Figures characterizing the best control law b_{best} derived thanks to `xMLC` with the $(P_{\text{Cros}}, P_{\text{Mut}}, P_{\text{Rep}}) = (0.5, 0.4, 0.1)$ combination.

A.3 Influence of population size and number of instructions

In this section, we present the influence of other parameters on the learning process of `xMLC`. In particular, we investigate the role of the population size $N_{\text{Pop.size}}$, the maximum number of instructions $N_{\text{Instr,max}}$ and the choice of the function library.

Parameter	Description	Value
	Function library	$F_1 = \{+, -, \times, \div\}$ $F_2 =$ $F_1 \cup \{\exp, \tanh, \sin, \cos, \log\}$
$\boldsymbol{s}$	Controller inputs	a_1, a_2
N_{Var}	Number of variable registers	3
N_{Cst}	Number of constant registers	3
$N_{\text{Instr,max}}$	**Max. number of instructions**	[2,5,10,20,50,100,200,500,1000]
$N_{\text{Pop.size}}$	**Population size**	[10,20,50,100,200,500,1000]
N_{G}	**Number of generations**	[100,50,20,10,5,2,1]
N_{Tour}	Tournament size	7
N_{E}	Elitism	1
P_{Cros}	Crossover probability	0.6
P_{Mut}	Mutation probability	0.3
P_{Rep}	Replication probability	0.1

Table A.6: Parameters for the parametric study of MLC. The studied parameters are in bold. Parameters are separated in two sets: (top) parameters that define the control laws, (bottom) parameters that define the learning process.

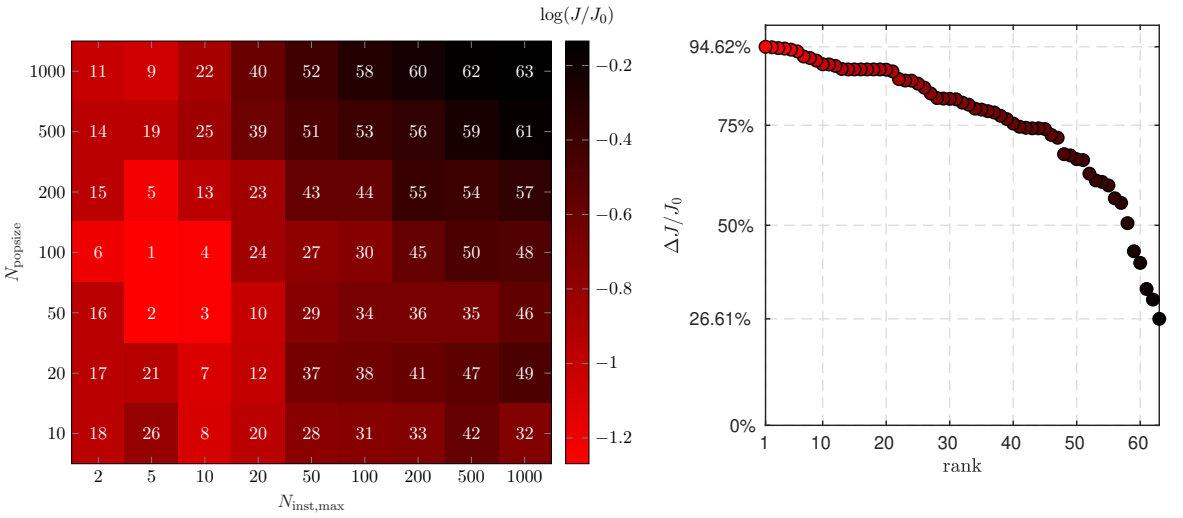
For a fair comparison between the parameters, we run all MLC optimizations with $N_i = 1000$ individuals. Seven population sizes are tested: 10, 20, 50, 100, 200, 500, 1000 with the adequate number of generations: 100, 50, 20, 10, 5, 2, 1. The 1000 individual run is equivalent to a Monte Carlo optimization, Also, we vary the maximum number of instructions $N_{\text{Instr,max}}$, nine values are tested: 2, 5, 10, 20, 50, 100, 200, 500, 1000. This means that for $N_{\text{Instr,max}} = 2$, the instruction matrix only contains two rows, and one thousand rows for $N_{\text{Instr,max}} = 1000$. Finally, we look at the impact of the function library in the optimization process. We investigate two libraries: $F_1 = \{+, -, \times, \div\}$ and $F_2 = \{+, -, \times, \div, \exp, \tanh, \sin, \cos, \log\}$. The rest of the parameters are the same as the previous section and are summarized in table A.6. The operator probabilities ($P_{\text{Cros}}, P_{\text{Mut}}, P_{\text{Rep}}$) are chosen according to Sec. A.2.

To have a relevant estimation of the performance of each combination of parameters, we realize $N_\rho = 100$ runs for each combination of parameters.

A.3.1 $F_1 = \{+, -, \times, \div\}$

Let's first look at the results for the library employed in Sec. A.2. In figure A.18, we notice, first, that there is a favored region in the parameter space for a best median performance. This region includes $N_{\text{Instr,max}} \in [5, 10]$ and $N_{\text{Pop.size}} \in [50, 100]$ and the performance decreases smoothly beyond this region. The best configuration of all is $(N_{\text{Instr,max}}, N_{\text{Pop.size}}) = (5, 100)$.

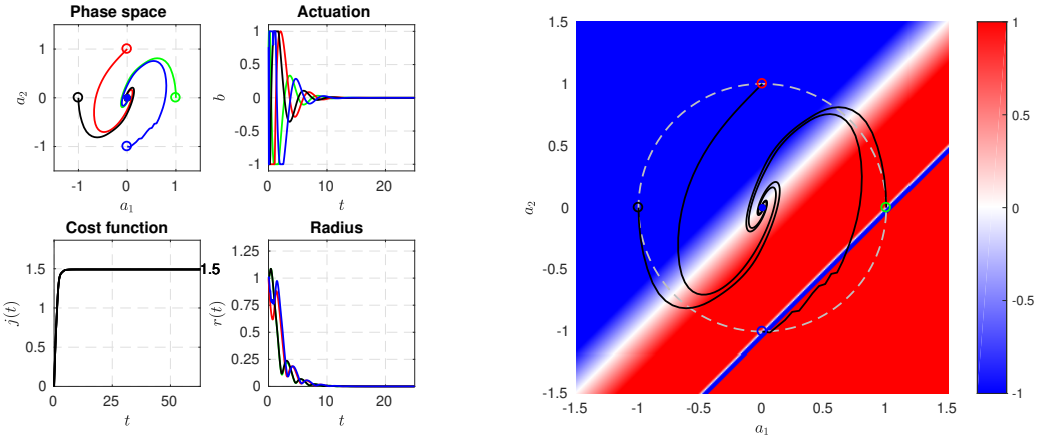
We remark that for a fixed population size, performance decreases when we increase the maximum number of instructions. This can be explained by the fact that as we allow more instructions, the search space becomes all the more bigger. Indeed, with more instructions, more and more complex control laws can be built, with the possibility to have a high level of function nesting. When a large number of instructions is employed, a bigger number of generations is preferred. Indeed, as the matrices have many more rows, several rounds of crossover and mutation are needed to shape the control law by selecting the adequate operators, registers and avoiding that further instructions destroy the intermediate expressions. This also shows that too much instructions can be a hindrance for the learning process. For the 'simple' problem that is the stabilization of the oscillator where rather simple control laws are expected to work, a small number of instructions seems to be favored. However, to chose too few instructions is not advantageous as we lose too much in complexity of the control laws. The maximum number of instructions should be large enough to include relevant solutions but not too much otherwise, the search space becomes too large to be effectively explored in a reasonable number of generations.



(a) (b)

Figure A.18: (a) Performance of MLC regarding population size and maximum number of instructions for the function library $F_1 = \{+, -, \times, \div\}$. The combination of parameters are ranked following the performance of their median realization. The combination ranked first is $(N_{\text{Instr,max}}, N_{\text{Pop.size}}) = (5, 100)$. (b) Cost reduction of the median realization of the different combinations. The horizontal axis represent the rank of the configuration described in (a). (a) and (b) share the same color code.

Concerning the population size, we notice that population sizes around 50 and 100 are favored. Also, some Monte Carlo samplings, especially with $N_{\text{Instr,max}} = 2$ and 5, perform better than MLC with $N_{\text{Pop.size}} = 10$. This is because small populations are more vulnerable to random variations of the population due to mutation; a given structure can more easily take over the population even if it is not the best one, discarding relevant structures and leading the population into a local minimum. This effect is also referred in biology as *genetic drift*.



(a) Phase space, actuation command, instantaneous cost function and radius.

(b) Visualization of the control law in the phase space. The limit cycle is depicted with a dashed line.

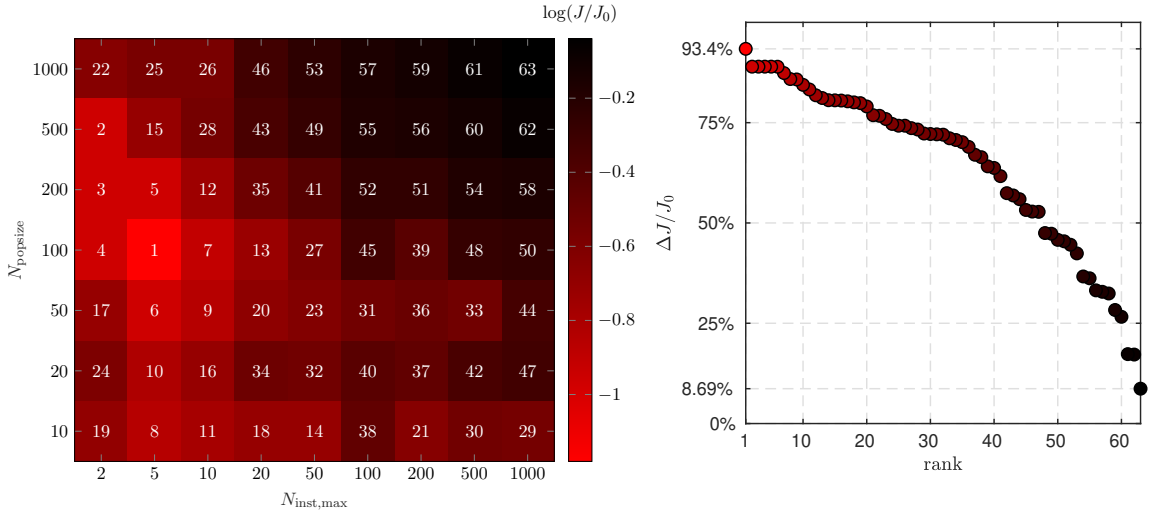
Figure A.19: Visualization of the controlled Landau oscillator with the best control law found for the function library $F_1 = \{+, -, \times, \div\}$.

The best control law among all realizations is:

$$b_{F_1,\text{best}} = \cfrac{(a_2-a_1)^2}{\cfrac{(a_2-a_1)}{-0.69096} - 5(a_2-a_1)} - 3(a_2-a_1)$$
$$\cfrac{(a_2-a_1)}{-0.69096} - 5(a_2-a_1) - 6(a_2-a_1)$$

and is depicted in figure A.19. This complex expression, nesting divisions, has been found with the combination $(N_{\text{Instr,max}}, N_{\text{Pop.size}}) = (20, 200)$ and reduces the cost by $\Delta J/J_0 = 95.50\%$. Even though a limit a 200 instructions was needed to find it, we note that the matrix of this control law has in fact 23 rows. Figure A.19 shows that the convergence of the oscillator towards the fixed point is not as fast as previous solution, indeed, we can still distinguish oscillations after the second period. However, this solution seems to favor less intense actuation and thus reducing the J_b component of the cost function. Indeed, the cost of the linear optimal control is as follows: $J_{a\,\text{opt}}/J_0 = 2.16\%$, $J_{b\,\text{opt}}/J_0 = 3.14\%$ and the cost of $b_{F_1,\text{best}}$ is: $J_a/J_0 = 1.76\%$, $J_b/J_0 = 2.74\%$

A.3.2 $F_2 = F_1 \cup \{\exp, \tanh, \sin, \cos, \log\}$



(a) (b)

Figure A.20: (a) Performance of MLC regarding population size and maximum number of instructions for the function library $F_2 = F_1 \cup \{\exp, \tanh, \sin, \cos, \log\}$. The combination of parameters are ranked following the performance of their median realization. The combination ranked first is $(N_{\text{Instr,max}}, N_{\text{Pop.size}}) = (5, 100)$. (b) Cost reduction of the median realization of the different combinations. The horizontal axis represent the rank of the configuration described in (a). (a) and (b) share the same color code.

We now add nonlinear functions in the function library, allowing oscillations with cos and sin, exponential and logarithm growth with exp and log and saturation with tanh. As we allow more complex expressions, we can expect to find complex and more efficient solutions than the linear optimal control. Figure A.20 show similar tendencies for the median realization as for the F_1 library. We note however that the performances have dropped in general. This can be explained by the fact, as we augment the function library, the search space becomes bigger and more individuals are needed to explore more control law structures. The combination $(N_{\text{Instr,max}}, N_{\text{Pop.size}}) = (5, 100)$ stands out from the other combinations as the best one with a reduction of $\Delta J/J_0 = 93.40\%$ of the unforced cost.

The best configuration for the F_2 library is $(N_{\text{Instr,max}}, N_{\text{Pop.size}}) = (5, 100)$ as it is ranked first in terms of median cost.

The all time best control law has been found for the combination $(N_{\text{Instr,max}}, N_{\text{Pop.size}}) = (20, 50)$ and reduces the cost by $\Delta J/J_0 = 97.08\%$ with a matrix containing 28 rows. We shall not show the control law as it is exceedingly complex. It comprises 215 instances of cos, 217 instances of $\div$, 216

instances of $(a_1 - a_2)$ and two times the same constant. Following the number of occurrences of the operators, we can assume that the same pattern has been repeated and nested to shape the control law. However, we cannot trust this solution as it appears to exploit the rather coarse time step chosen for integration. Indeed, the dynamics and cost of this control law vary considerably when the integration time step decreases.

The success of MLC is that it managed to detect and exploit the most receptive state of the oscillator to efficiently control it with minimum actuation energy. The most difficult task for MLC may have been to build the transient part of the control, to guide the system towards the state of interest. Indeed, the best strategy found by MLC has been to let the system evolve freely until it gets to the region where the forcing is the most efficient, see figure A.17. Such strategy balances effectively stabilization and actuation power. It has been built thanks to nonlinear functions and we can assume that augmenting the function library with more nonlinear functions, such as inequality signs ($>$ or $\leq$) and boolean operators, can help the learning of better control laws. However, one must select the function library carefully since a too diverse library lowers the general performance.

Appendix B

Define user's problem

The definition of a MLC problem and its parameters is done by two files: `MyProblem_problem.m` and `MyProblem_parameters.m` respectively. In Sec. B.1, the `Plant/MyPlant/MyProblem_parameters.m` file is detailed. This file should be used as a template to build one's own parameter file. A template for the definition of a MATLAB problem is also given in Sec. B.2. To adapt the parameter file to her/his own problem, the user is invited to create a new folder in `Plant/` with a name related to her/his problem. There is no restriction on the name of the folder as it is not used in the code.

B.1 User's parameter file

The `Plant/MyPlant/MyProblem_parameters.m` is a basis to define one's own parameter file. In this section, we provide comments and suggestions on how to choose the parameters. The parameters displayed are just examples, they are coherent between them but do not represent any real problem.

```
function parameters = MyPlant_parameters()
        % Template file to write one's own parameter file.
        % Every instance of "MyPlant" should be replaced by an adequate name.
        % Lines ending with a "*" should not modified.
        %
        % Guy Y. Cornejo Maceda, 2022/07/01
        %
        % See also default_parameters.

        % Copyright: 2022 Guy Cornejo Maceda (gy.cornejo.maceda@gmail.com)
        % The MIT License (MIT)

%% Options
        parameters.verbose = 2;
```

The parameter file is divided in four main sections. First, the general parameters define the type of problem and how the `xMLC` code handles it.

```
%% General parameters
    parameters.Name = 'TestRun'; % Name used to save the MLC object.
    % Name of the problem file. This name corresponds to the first part of
    % the MyPlant_problem.m file.
    % Of course, one can have several parameter files referring to the same
    % problem.
    parameters.EvaluationFunction = 'MyPlant';
    % Following if the plant is a MATLAB file or an external
    % solver/experiment:
    %       - 'MATLAB'  : when there is a *_problem.m file.
    %       - 'external': should be used when a numerical solver is
    %       employed.
    %       - 'LabView' : should be used when xMLC is interfaced with an
    %       experiment using LabView.
    % For the last options, the EvaluationFunction parameter does not
    % matter.
```

```
32      parameters.ProblemType = 'MATLAB'; % 'MATLAB', 'external' or 'LabView'
33      % Path for external evaluation.
34      % When the code is interfaced with an external solver or LabView,
35      % files are exchanged via the folder in the PathExt path.
36      % For 'MATLAB' type problems, this path is useless.
37      parameters.PathExt = '/Costs'; % Folder to exchange information.
```

Secondly, the problem parameters define the inputs and outputs of the controller, and some parameters related to the system to control.

```
39 %% Problem parameters
40      % Inputs and outputs definition
41          % The inputs and outputs are considered from the controller point
42          % of view. Thus, outputs corresponds to the actuation commands and
43          % the inputs are the sensors signals or time dependent functions.
44          % Outputs---Control laws
45          ProblemParameters.OutputNumber = 2; % Number of control laws
46          % Inputs---Sensors and time dependent functions
47              % si(t)---Sensor signals
48              ProblemParameters.NumberSensors = 2;
49              % Name of the variables in the solver.
50              % Here 'a1' and 'a2' represent state vectors.
51              ProblemParameters.Sensors = {'a1','a2'};
52              % hi(t)---Time dependent functions
53              ProblemParameters.NumberTimeDependentFunctions = 2;
54              % Include the list of time dependent functions in the
55              % MATLAB/Octave syntax. The time variable should be 't'.
56              ProblemParameters.TimeDependentFunctions = ...
57                  {'sin(2*pi*1.5*t)','sin(2*pi*2.5*t)'};
58              % Include the list of time dependent functions one more time in
59              % the solver's syntax (Fortran, MATLAB, LabView, ...).
60              % If no time dependent functions are employed then comment.
61              ProblemParameters.TimeDependentFunctions(2,:) = ...
62                  {'sin(2*pi*1.5*t)','sin(2*pi*2.5*t)'};
63          % Consistency test.                                              %*
64          ProblemParameters.InputNumber = ...                              %*
65              ProblemParameters.NumberSensors+...                          %*
66              ProblemParameters.NumberTimeDependentFunctions;              %*
67          % Control syntax definition.                                     %*
68          Sensors = cell(1,ProblemParameters.NumberSensors);               %*
69          TDF = cell(1,ProblemParameters.NumberTimeDependentFunctions);    %*
70          for p=1:ProblemParameters.NumberSensors                          %*
71              Sensors{p} = ['s(',num2str(p),')'];                          %*
72          end                                                              %*
73          for p=1:ProblemParameters.NumberTimeDependentFunctions           %*
74              TDF{p} = ['h(',num2str(p),')'];                              %*
75          end                                                              %*
76          ControlSyntax = horzcat(Sensors,TDF);                            %*
```

The plant parameters are divided in two groups: The parameters that are essential because used in the code, and additional parameters employed in the problem file. The user is free to add any parameters she/he wants it this part. All parameters are sent to the problem for the evaluation of one individual.

```
99      % Plant parameters.
100         % Essential problem parameters used in the code.
101             ProblemParameters.T0 = 0; % Not always used
102             ProblemParameters.Tmax = 10; %
103             % Actuation limitation : [lower bound,upper bound]
104             ProblemParameters.ActuationLimit = ...
105                 [-2,2;... % Boundaries for the first actuator.
106                 -1,1];    % Boundaries for the second actuator.
107             % The evaluation of the cost function and the control points
108             % will be rounded to the 'RoundEval' parameter.
109             ProblemParameters.RoundEval = 6;
110             % Estimate performance
```

```
            % When a control law is evaluated several times, its
            % performance can be computed by taking the average of all the
            % evaluations.
            % Other options are also possible if there is a drift, such as:
            %   - 'last' : takes the cost of the evaluation
            %   - 'worst': takes the cost of the worst evaluation
            %   - 'best' : takes the cost of the best evaluation
            ProblemParameters.EstimatePerformance = 'mean';

        % Problem parameters used in the *_problem.m file.
            % Maximum evaluation time otherwise returns a bad value
            ProblemParameters.TmaxEv = 5;
            % Penalization parameter
            % It is a vector that multiplies the secondary components of
            % the cost function such as:
            % J = Ja + gamma(1)*Jb + gamma(2)*Jc
            ProblemParameters.gamma = [0.01,10];
            % Other parameters used by the solver or normalization of the
            % cost function.
            ProblemParameters.ReynoldsNumber = 100; % For example.
            ProblemParameters.J0 = 1;               % For example.
            ProblemParameters.Jmin = 0;             % For example.
            ProblemParameters.Jmax = inf;           % For example.
            % ...

    % Assignation
    parameters.ProblemParameters = ProblemParameters; %*
```

Then, the control law parameters define the space of the matrices, i.e., the space of control laws. The following parameters define the maximum number of lines (or instructions) in the matrix, the range of each column, i.e., the mathematical operations, and the registers (constant and variable).

```
%% Control law parameters
    % Number of instructions.
    % InitMin and InitMax define the range of number of instructions when
    % the matrices are generated randomly.
    % The number of instructions for each matrix is then chosen randomly
    % with an uniform distribution.
    % Max defines the maximum number of instructions during the learning
    % process. This is used during the crossover operation as the number of
    % instructions can grow larger than the initial number.
        ControlLaw.InstructionSize.InitMax=20;
        ControlLaw.InstructionSize.InitMin=1;
        ControlLaw.InstructionSize.Max=20;

    % Operators.
        ControlLaw.OperatorIndices = [1:5,7:9];
            %  implemented:  - 1  addition       (+)
            %                - 2  subtraction    (-)
            %                - 3  multiplication (*)
            %                - 4  division       (%) - protected (my_div)
            %                - 5  sinus          (sin)
            %                - 6  cosines        (cos)
            %                - 7  logarithm      (log) - protected (my_log)
            %                - 8  exp            (exp)
            %                - 9  tanh           (tanh)
            %                - 10 square         (.^2)
            %                - 11 modulo         (mod)
            %                - 12 power          (pow)
        % Precision of the evaluation of the control laws.
        % The default value is 6.
        % If this value is changed, one must also update the protected
        % operators: my_div and my_log.
        ControlLaw.Precision = 6;

```

```
151     % Definition of the registers.
152         % Number of variable registers.
153         % The minimum number is the sum between the number of inputs and
154         % outputs.
155         VarRegNumberMinimum = ProblemParameters.OutputNumber+...          %*
156             ProblemParameters.InputNumber;                                %*
157         % More variable registers can be added if necessary.
158         ControlLaw.VarRegNumber = VarRegNumberMinimum + 3;
159         % Number of constant registers.
160         % Here, we only set random constants in the constant registers but
161         % one can also include sensor signals.
162         ControlLaw.CstRegNumber = 4;
163         % Range of the random constants constants.
164         % In this example, the range of the four constants is [-1;1].
165         ControlLaw.CstRange = repmat([-1,1],ControlLaw.CstRegNumber,1);
166         % Total number of registers.
167         ControlLaw.RegNumber = ...          % Total number of operands     *
168             ControlLaw.VarRegNumber + ...   % Number of variable registers *
169             ControlLaw.CstRegNumber;        % Number of constant registers *
170         % Register initialization (recommended)
171             NVR = ControlLaw.VarRegNumber;
172             RN = ControlLaw.RegNumber;
173             r{RN}='0';
174             r(:) = {'0'};
175             % Variable registers
176             for p=1:ProblemParameters.InputNumber
177                 r{p+ProblemParameters.OutputNumber} = ControlSyntax{p};
178             end
179             % Constant registers
180             minC = min(ControlLaw.CstRange,[],2);
181             maxC = max(ControlLaw.CstRange,[],2);
182             dC = maxC-minC;
183             for p=NVR+1:RN
184                 r{p} = num2str(dC(p-NVR)*rand+minC(p-NVR));
185             end
186         % Assignation
187         ControlLaw.Registers = r;                                         %*
```

To accelerate the learning, individuals with originally different mathematical expression but exactly equal after simplification are searched to be removed. For this, the control laws are evaluated on random sensor samplings. Those samplings are referred as 'control points' in the code. Random time samples are also generated for the time dependent functions.

```
189         % Control law estimation.
190         ControlLaw.ControlPointNumber = 1000;
191         % Sensor range to define the sampling points.
192         % In this case, the range of all sensors is [-2,2].
193         ControlLaw.SensorRange = repmat([-2 2],...
194             ProblemParameters.NumberSensors,1);
195         % Definition of random sampling points (ControlPoints) and random
196         % time samplings for the control law estimation
197             Nbpts = ControlLaw.ControlPointNumber;                        %*
198             Rmin = min(ControlLaw.SensorRange,[],2);                      %*
199             Rmax = max(ControlLaw.SensorRange,[],2);                      %*
200             dR = Rmax-Rmin;                                               %*
201         ControlLaw.EvalTimeSample = rand(1,Nbpts)*ProblemParameters.Tmax;%*
202         ControlLaw.ControlPoints = ...                                    %*
203             rand(ProblemParameters.NumberSensors,Nbpts).*dR+Rmin;         %*
204
205     % Assignation
206     parameters.ControlLaw = ControlLaw;                                   %*
```

Finally, the MLC parameters define the optimization process. In addition to the genetic operators probabilities parameters, there are also some screening of the individuals options.

```
%% MLC parameters
    % Population size
    parameters.PopulationSize = 10;
    % Optimization options
        % Optimization of the first generation only (Monte Carlo).
        % It removes the duplicated individuals, redundant, etc.
    parameters.OptiMonteCarlo = 1;
    % Remove individuals whose evaluation failed (bad)
    parameters.RemoveBadIndividuals = 1;
    % Remove already evaluated individuals
    parameters.RemoveRedundant = 1;
    % Remove the individuals if they have already been evaluated in a
    % earlier generations.
    parameters.CrossGenRemoval = 1;
    % Evaluate the initial condition of the registers (here:b=0)
    parameters.ExploreIC = 1;
    % The crossover and mutation operators are until new individuals are
    % generated or MaxIterations iterations is reached.
    parameters.MaxIterations = 100;
    % Reevaluation of the individuals for stochastic problems.
    % This parameter implicitly set to 1 for experiments.
    %   - 0: No revaluation
    %   - 1: Force the reevaluation of the replicated and "elitism" indiv.
    %   - n: Each individual is reevaluated n times.
    parameters.MultipleEvaluations = 0;
    % Selection parameters.
    parameters.TournamentSize = 7;
    parameters.p_tour = 1;
    % Genetic operator parameters
    parameters.Elitism = 1;
    parameters.CrossoverProb = 0.6;
    parameters.MutationProb = 0.3;
    parameters.ReplicationProb = 0.1;
    % Other genetic parameters
        % How many mutations per matrix (MutationType)?
        %   - 'classic': depends on the mutation rate. The mutation rate is
        %   the probability that a given line changes.
        %   - 'at_least_one': the mutation rate is set such as there is at
        %   least one mutation per matrix.
        %   - 'number_per_chromosome': the mutation rate is set such as
        %   there is at least MutationNumber mutations per matrix.
        %   least one mutation per matrix.
    parameters.MutationType = 'number_per_matrix';
    parameters.MutationNumber = 1;
    parameters.MutationRate = 0.05;
    % Number of cuts in the matrices for the crossover operation.
    parameters.CrossoverPoints = 1;
    % If CrossoverMix=1, the offsprings are built by alternating their
    % parents sections, if CrossoverMix=0, the sections are selected
    % randomly.
    parameters.CrossoverMix = 1;
    % The crossover operation gives two offsprings by crossover operation.
    parameters.CrossoverOptions = {'gives2'};
    % Other parameters
    % Cost value for individuals whose evaluation failed.
    parameters.BadValue = 10^36;
    % Remove the individuals that do not satisfy a given test.
    % The removed individuals are referred as "wrong" individuals.
    parameters.Pretesting = 0;

%% Constants
    parameters.PHI = 1.61803398875;

%% Other parameters
```

```
    % Parameter that stores the list of saving times.
    % Can help to keep track of the latest version.
    parameters.LastSave = '';

end
```

B.2 User's MATLAB problem file

In this section, we describe the typical structure of MATLAB problem file. We detail in particular, the types of the inputs and outputs.

The input of the function, `Arrayb`, is a $N_\mathrm{b} \times 1$ 'cell' array, where N_b is the number of components of the control law. The elements of the array are string of characters that define the control law.

```
function  J_out = MyPlant_problem(Arrayb,parameters,visu)
  % MYPLANT_PROBLEM is a template file to define one's own problem file to
  % be solved with MATLAB.
  % The user can also include a calling to his solver in this file.
  % Several initial conditions can be used.
  % To help understand the commands, we illustrate the commands with
  % Arrayb set to {'1.34','a1+a2*sin(2*pi*1.5*t)'}, for example.
  % This example could correspond to the case where the individual is:
  % {'1.34','s(1)+s(2)*h(1)'}.
  % The values of s(1), s(2) and h(1) are replaced before the evaluation by
  % the values set in the parameter file, see MyPlant_parameters.m
  %
  % Guy Y. Cornejo Maceda, 2022/07/01
  %
  % See also LandauOscillator_problem.

  % Copyright: 2022 Guy Cornejo Maceda (gy.cornejo.maceda@gmail.com)
  % The MIT License (MIT)

%% Parameters
    ActuationLimit = parameters.ProblemParameters.ActuationLimit;
    ActMin = ActuationLimit(:,1);
    ActMax = ActuationLimit(:,2);
    gamma = parameters.ProblemParameters.gamma;
    BadValue = parameters.BadValue;
    RoundEval = parameters.ProblemParameters.RoundEval;

%% Control law synthesis
    % Bound the actuation with a clip function.
    BoundArrayb = limit_to(Arrayb,ActuationLimit);
    % The results of this command is then:
    %   2x1 cell array
    %
    %     'clip(1.34,-2,2)'
    %     'clip(a1+a2*sin(2*pi*1.5*t),-1,1)'

    % Control law
    % The i-th control law is the i-th element of Arrayb.
    bx1=BoundArrayb{1};
    bx2=BoundArrayb{2};
    % Definition
    eval(['b = @(t,a1,a2)[',bx1,';',bx2,'];']);

%% Objective
% Stabilization of the fixed point (example)

%% Some problem parameters
    % Parameter A
    A = 1;
    % Parameter B
```

```
    B = 1;
    % Initial conditions
    initial_conditions = [0;0.1];

%% Resolution parameters
    % Solver
        solver = 'ode5';
    % Time discretization
        % Number of time steps
        N = 500;
        T0 = parameters.ProblemParameters.T0;
        Tmax = parameters.ProblemParameters.Tmax; % frequency=1
        time = linspace(T0,Tmax,N+1);
        TmaxEv = parameters.ProblemParameters.TmaxEv;
```

In this example, a solver with fixed size step is employed to integrate the ODE, however when a solver with a variable size step is chosen (like `ode45`) the integration time can significantly increase. Thus, we add a function `T_maxevaluation` to limit the integration time. This function returns an error when the integration time is larger than the variable `TmaxEv`, triggering the exception in the `try` function.

```
%% Equation resolution
    % Equations
    % Unforced dynamical system (example)
     DynSys = @(t,a)[(A-a(1).^2-a(2).^2).*a(1)-B*a(2);...
                     (A-a(1).^2-a(2).^2).*a(2)+B*a(1)];

    % Controlled dynamical system
        ConDynSys = @(t,a) (DynSys(t,a) + b(t,a(1),a(2)) +...
        T_maxevaluation(TmaxEv,toc)*[0;0]);

%% Time integration
% A try function is employed to treat the case where the evaluation fails.
try
    % Resolution
        tic
        y = feval(solver,ConDynSys,time,initial_conditions);

%% Cost function
    % Ja (example)
    ja = y(:,1).^2+y(:,2).^2;
    Ja = mean(ja);
    % Jb (example)
    % Evaluate the control law -> cell structure
    b_cell = arrayfun(b,time',y(:,1),y(:,2),'UniformOutput',false);
    % Convert in matrix
    b_matrix = cell2mat(transpose(b_cell));
    jb = sum(b_matrix.^2,2);
    Jb = mean(jb);
    % Jc (example)
    Jc = rand;
    J = Ja+gamma(1)*Jb+gamma(2)*Jc;

catch err
    J_out = {BadValue,BadValue,BadValue,BadValue};
    fprintf(err.message);
    fprintf('\n');
    return
end
```

The output, `J_out`, is also a cell array. Its dimension is the number of components of the cost function +1. It this example, J is composed of three elements, thus the output is: J, J_a, J_b, J_c. A cell type is chosen for the output to allow cost function of different types (boolean, string of characters, etc.).

```
%% Output
    J_round = round(J,RoundEval);
```

```
107     Ja_round = round(Ja,RoundEval);
108     Jb_round = round(Jb,RoundEval);
109     Jc_round = round(Jc,RoundEval);
110     J_out = {J_round,Ja_round,Jb_round,Jc_round};
111 
112 %% Plot
113 if nargin > 2 && visu
114    figure
115    % Include your figure here.
116 end
117 end
```

Appendix C

Interfacing with an external numerical solver or an experiment

In this appendix, we give some guidance on how to interface `xMLC` with an external numerical solver or an experiment.

C.1 External numerical solver

For external numerical solver, e.g., based on Fortran or Python, there is mainly two ways to interface the codes. If the evaluation time of each individual takes a few seconds, the simplest way is to call the solver directly from the problem file along with the adequate parameters. When the computation of one individual takes several dozens of minutes or even a few hours in a single processor, it is recommended to evaluate the individuals *in parallel* on a computer cluster. `xMLC` includes several scripts that help the management of the control laws to evaluate and the cost files once the computation is over.

The first script is `MLC_tools/External_evaluation_START.m`:

```
    % EXTERNAL_EVALUATION_START starts the run.
    % This script generates the first generation of individuals and
    % creates a file Gen1population.mat , a cell array containing the
    % control laws to evaluate.
    % The file is locate in save_runs/RUNNAME/Populations.
    %
    % Guy Y. Cornejo Maceda, 2022/07/01
    %
    % See also external_evaluation_CONTINUE, External_evaluation_END.

    % Copyright: 2022 Guy Cornejo Maceda (gy.cornejo.maceda@gmail.com)
    % The MIT License (MIT)

Initialization;
%% Start
    mlc=MLC('MyExternalPlant');
    mlc.parameters.PathExt='';

%% Generate population
    mlc.generate_population;

%% Save
    mlc.save_matlab('Gen0');
```

This script creates a MLC object with the 'MyExternalPlant' problem. This problem is similar to the 'MyPlant' one, except that the `Name` and `ProblemType` parameters are set to `ExternalTestRun` and `'external'`, respectively. The script creates a MAT file (Gen1population.mat) containing the control laws to be evaluated in cell array. This file is located in `save_runs/MyExternalPlant/` in this example. The script also saves the MLC object generated under the name `Gen0`. The index '0' indicates that the first generation has not been evaluated yet. Once this script is executed, the evaluation of the individuals on a computer cluster can start.

The expected output of each individual evaluation is a file containing the components J_k of the cost function. The files should be in ASCII format and named `GenXIndY.dat` where `X` is the generation number and `Y` the creation order of the individual. Here is an example of such file for the evaluation of the 14^{th} individual of the 2^{nd} generation.

```
1.345 2.050 0.767
```

All these files should be located in the folder defined by the variable `PathExt`; its value is modified on line 17 of `External_evaluation_START.m`. No cost file is expected for individuals whose evaluation failed.

Once the evaluation the evaluation of all individuals is done, to continue the optimization process, one needs to run the function `MLC_tools/External_evaluation_CONTINUE.m` with the generation that has been evaluated.

```
1  function External_evaluation_CONTINUE(gen)
2      % EXTERNAL_EVALUATION_CONTINUE continues the run.
3      % To be used after the evaluation of the individuals of generation GEN.
4      % Retrieves the cost information and makes the population evolve.
5      % New control laws are generated in the Population folder and ready to
6      % be evaluated.
7      %
8      % Guy Y. Cornejo Maceda, 2022/07/01
9      %
10     % See also External_evaluation_END, External_evaluation_START.
11
12     % Copyright: 2022 Guy Cornejo Maceda (gy.cornejo.maceda@gmail.com)
13     % The MIT License (MIT)
14
15 %Evolve_population_script
16 Initialization;
17 mlc=MLC('MyExternalPlant');
18 %% Load
19     mlc.load_matlab('ExternalTestRun',['Gen',num2str(gen-1)]);
20
21 %% Complete
22     matJ = External_build_matJ(mlc.parameters,gen);
23     complete_evaluation(mlc,gen,matJ);
24
25 %% Evolve
26     evolve_population(mlc);
27
28 %% Save
29     mlc.save_matlab(['Gen',num2str(length(mlc.population)-1)]);
```

The script loads the previous save of the MLC class object, loads the cost files, generates the next generation of individuals and create the associated MAT file containing the control laws. The function that loads the cost files is `External_build_matJ` (line 22), its output is a matrix containing the components of the cost function (the columns) for each individual (the rows). One is free to adapt the code and especially `External_build_matJ` to her/his own file system. The function `External_evaluation_CONTINUE` needs then to be iterated to advance the generations.

For the final generation the `External_evaluation_END` function needs to be run with the final generation as input. `External_evaluation_END` does exactly the same as `External_evaluation_CONTINUE` except generating the next generation. The optimization process can then be resumed by running the `External_evaluation_CONTINUE` function.

In practice, these three scripts ('START', 'CONTINUE' and 'END') are meant to be ran directly at the start of MATLAB. Once they are executed, the MATLAB session can be ended. The best way to use them is in shell script comprising all the commands. Here is an example of such script in bash:

```
#!/bin/bash

# Parameters
GenMax=10

# --- Start ---
```

```
matlab -r "External_evaluation_START;"
# Evaluation
# ...

# Initialization of the generation counter
Gen=1

# --- Optimization loop ---
while [ "$Gen" != "GenMax" ]
do
        matlab -r "External_evaluation_CONTINUE($Gen)"
        # Evaluation
        # ...

        # Increment Gen
        Gen=$(($Gen+1))
done

# --- Complete the final generation
matlab -r "External_evaluation_END($Gen)
```

C.2 Experiment

The `xMLC` code, running on MATLAB, has been interfaced with several experiments via LabVIEW and dSPACE/Simulink-based platforms. For an experiment, there are essentially two main loops:

- The fast evaluation loop, that runs at the sampling frequency. It contains the plant, sensors, actuators and real time controller.
- The slow learning loop, that updates the control laws once they are evaluated. Its characteristic frequency is much lower than the evaluation loop. It contains the real time controller and the MATLAB session.

The first step to interface the code with an experiment is to be able to run a real-time control loop for any control law or mathematical expression. Here are some elements the user needs to take into account to run a real-time control loop and to ease the interface with `xMLC` :

- The graphic model of the experiment shall include a clip function for the actuation command sent to the actuators. This assures the actuators are not harmed by out of range inputs.
- The sampling frequency is well chosen such as the frequencies of interest are well resolved. The Nyquist theorem requires the sampling frequency to be at least twice the largest frequency.
- Automatic saving of the time series under a unambiguous.

Once any control law can be evaluated, there are essentially two ways to interface the `xMLC` with the experiment:

- Update the set of control law at each generations. This approach has been employed when interfacing `xMLC` with dSPACE/Simulink. The `external` option needs to be chosen as `ProblemType` in the parameters. The idea is to have a Simulink model including a list of control laws that is compiled and sent to the real time controller. The list of control laws is a function that takes as input the sensor signals, an integer and a gives back the actuation command as output. The integer is used to 'select' the control law in the list. In `xMLC` , this function can be automatically generated with the method `@MLC/expe_create_control_select.m`. The function is named `ControlLawSelect.m` and is located in `save_runs/tmp/`. This approach needs a recompilation of the Simulink once every generation. Once, the generation is evaluated and the time series saved, the cost of each individual can be computed and sent to `xMLC` to create the generation. This approach is the more complex as it requires an extra Python or Bash layer to command the real time controller and the MATLAB session.

- Another approach is to update the control laws one by one. This approach has been employed when interfacing with a LabVIEW system. The `LabView` option needs to be chosen as `ProblemType` in the parameters for this approach. Instead of sending the control law to a MATLAB function, `xMLC` creates a script, `LabViewControlLaw.txt`, containing the control law and readable by LabVIEW. The `LabViewControlLaw.txt` file is created in the external path defined by `PathExt` in the parameters. This approach does not require any compilation as it employs a control law parser. Once the evaluation is done and the time series saved, the costs can be computed and send to `xMLC`. The process will then continue automatically. This approach is the most simple but an extra step of control law simplification may be needed if the parsing takes too much time compared to sampling time. The user is free to modify:
  - The `MLC_tools/CreatefunctionLabview.m` function to create the `LabViewControlLaw.txt` file that suits her/his needs;
  - The `@MLCind/evaluate_ind.m` method in the 'LabView' section to send the control law to LabVIEW and compute the cost from the time series.

We shall not detail the interfacing any longer as it often depends on the experiment. For more information on the interfacing of the code with LabVIEW, dSPACE/Simulink or any other platform please contact Guy Y. Cornejo Maceda at: Yoslan@hit.edu.cn.

Appendix D

Accelerating the learning

In this appendix, we describe some accelerators that manage to increase the learning rate of MLC. The main idea of these accelerators is to avoid redundant evaluations, meaning to prevent the evaluation of the same control laws. First, we need to detect equivalent control laws. However, this is not an easy task as simplification of two mathematical expressions is not guaranteed to give the same results because of the commutative operators. Moreover, another complication is the protection of the $\div$ and log operators. Because of the protection, these functions present a discontinuous behavior near 0 that prevents from simplification. Thus, we propose another way to detect equivalent expressions. To detect if two control laws are similar, we evaluate them on random sample inputs and we compare the control outputs. These methods allows to detect mathematical equivalent control laws even though their expressions are very different. Indeed, thanks to such test, control laws that are beyond the actuation thresholds will also be ruled out. For example, the control laws $b = 1.01$ and $b = 10^23$ will be considered as the same control laws if the threshold is set to 1. Moreover, the random samples are taken in the range of the sensors signals making the comparison between the control laws even more meaningful. In practice, each time a new control law is built, we evaluate it over 1000 random sample inputs and store the result in the database. Each new control law is then compared to the database to verify if it has not been already evaluated.

Equivalent control laws can then be filterer out in two ways:

In the population: if a new individual is equivalent to an individual already present in the population, then it is discarded and a new individual is built (randomly if it is the first generation or with one of the genetic operators).

In the database: if a new individual is equivalent to a previous individual in the current or past generations, then it is discarded and a new individual is built.

Of course the second option includes the first one. Those rules do not apply for individuals generated thanks to replication and elitism as their role is to emulate memory through the generations. Those filtering assures that the population of individuals is always moving towards unexplored regions of the control landscape and backward steps are not possible.

Bibliography

S. Asai, H. Yamato, Y. Sunada, and K. Rinoie. Designing machine learning control law of dynamic bubble burst control plate for stall suppression. In *2019 AIAA SciTech Forum*, San Diego, CA, 2021. Paper 1899.

D. Barros, J. Borée, B. R. Noack, A. Spohn, and T. Ruiz. Bluff body drag manipulation using pulsed jets and Coanda effect. *J. Fluid Mech.*, 805:442–459, 2016.

P. W. Bearman. The effect of base bleed on the flow behind a two-dimensional model with a blunt trailing edge. *Aeronautical Quarterly*, 18(03):207–224, 1967.

N. Benard, J. Pons-Prats, J. Periaux, G. Bugeda, P. Braud, J.P. Bonnet, and E. Moreau. Turbulent separated shear flow control by surface plasma actuator: Experimental optimization by genetic algorithm approach. *Exp. Fluids*, 57(2):22:1–17, 2016.

A. Blanchard, G. Y. Cornejo Maceda, D. Fan, Y. Q. Li, Y. Zhou, B. R. Noack, and T. Sapsis. Bayesian optimization of active flow control. *Acta Mechanica Sinica*, 37(1):47–52, 2022.

M. Brameier and W. Banzhaf. *Linear Genetic Programming*. Springer Science & Business Media, 2006.

S. L. Brunton and B. R. Noack. Closed-loop turbulence control: Progress and challenges. *Appl. Mech. Rev.*, 67(5):050801:01–48, 2015.

S. L. Brunton, B. R. Noack, and P. Koumoutsakos. Machine learning for fluid mechanics. *Ann. Rev. Fluid Mech.*, 52:477–508, 2020.

R. Castellanos, G. Y. Cornejo Maceda, I. de la Fuente, B. R. Noack, A. Ianiro, and S. Discetti. Machine learning flow control with few sensor feedback and measurement noise. *Physics of Fluids*, 34(4): 047118:1–17, 2022.

W. Chen, C. Ji, Md M. Alam, J. Williams, and D. Xu. Numerical simulations of flow past three circular cylinders in equilateral-triangular arrangements. *Journal of Fluid Mechanics*, 891:1–44, 2020.

C. Chovet, L. Keirsbulck, B. R. Noack, M. Lippert, and J.-M. Foucaut. Machine learning control for experimental shear flows targeting the reduction of a recirculation bubble. In *The 20th World Congress of the International Federation of Automatic Control (IFAC)*, pages 1–4, Toulouse, France, 2017.

G. Y. Cornejo Maceda. *Gradient-enriched machine learning control exemplified for shear flows in simulations and experiments*. PhD thesis, Université Paris-Saclay, 2021.

G. Y. Cornejo Maceda, Noack B. R., F. Lusseyran, N. Deng, L. Pastur, and M. Morzyński. Artificial intelligence control applied to drag reduction of the fluidic pinball. *Proc. Appl. Math. Mech.*, 19(1): e201900268:1–2, 2019.

G. Y. Cornejo Maceda, Y. Li, F. Lusseyran, M. Morzyński, and B. R. Noack. Stabilization of the fluidic pinball with gradient-based machine learning control. *J. Fluid Mech.*, 917:A42:1–43, 2021.

A. Debien, K. A. F. F. von Krbek, N. Mazellier, T. Duriez, L. Cordier, B. R. Noack, M. W. Abel, and A. Kourta. Closed-loop separation control over a sharp-edge ramp using genetic programming. *Exp. Fluids*, 57(3):40:1–19, 2016.

N. Deng, B. R. Noack, M. Morzyński, and L. R. Pastur. Low-order model for successive bifurcations of the fluidic pinball. *J. Fluid Mech.*, 884:A37, 2020.

J. C. Doyle, B. A. France, and A. R. Tannebaum. *Feedback Control Theory*. CRC Press, 1992.

D. C. Dracopoulos. *Evolutionary Learning Algorithms for Neural Adaptive Control*. Springer-Verlag, London, etc., 1997.

T. Duriez, V. Parezanović, J.-C. Laurentie, C. Fourment, J. Delville, J.-P. Bonnet, L. Cordier, B. R. Noack, M. Segond, M. W. Abel, N. Gautier, J.-L. Aider, C. Raibaudo, C. Cuvier, M. Stanislas, and S. Brunton. Closed-loop control of experimental shear layers using machine learning (invited). In *7th AIAA Flow Control Conference*, pages 1–16, Atlanta, Georgia, USA, 2014.

T. Duriez, S. L. Brunton, and B. R. Noack. *Machine Learning Control — Taming Nonlinear Dynamics and Turbulence*, volume 116 of *Fluid Mechanics and Its Applications*. Springer-Verlag, 2017.

Y. El-Sayed, P. Oswald, S. Sattler, K. Pradeep, R. Radespiel, C. Behr, M. Sinapius, J. Petersen, P. Wierach, M. Quade, M. Abel, B. R. Noack, and R. Semaan. Open- and closed-loop control investigations of unsteady Coanda actuation on a high-lift configuration. In *AIAA Aviation*, pages 1–13, Atlanta, Georgia, USA, 2019. AIAA 2018-3684.

D. Fan, L. Yang, Z. C. Wang, M. S. Triantafyllou, and G. M. Karniadakis. Reinforcement learning for bluff body active flow control in experiments and simulations. *Proc. Natl. Acad. Sci. USA*, 117(42): 26091–26098, 2020.

G. Flügel. Ergebnisse aus dem Strömungsinstitut der Technischen Hochschule Danzig. In *Jahrbuch der Schiffbautechnischen Gesellschaft*, pages 87–113. Springer, 1930.

N. Gautier, J.-L. Aider, T. Duriez, B. R. Noack, M. Segond, and M. W. Abel. Closed-loop separation control using machine learning. *J. Fluid Mech.*, 770:424–441, 2015.

D. Geropp. Process and device for reducing the drag in the rear region of a vehicle, for example, a road or rail vehicle or the like. United States Patent **US 5407245 A**, 1995.

D. Geropp and H.-J. Odenthal. Drag reduction of motor vehicles by active flow control using the Coanda effect. *Exp. Fluids*, 28(1):74–85, 2000.

A. Hervé, D. Sipp, P. J. Schmid, and M. Samuelides. A physics-based approach to flow control using system identification. *J. Fluid Mech.*, 702:26–58, 2012.

R. Ishar, E. Kaiser, M. Morzynski, M. Albers, P. Meysonnat, W. Schröder, and B. R. Noack. Metric for attractor overlap. *J. Fluid Mech.*, 874:720–752, 2019.

J. Kim and T.R. Bewley. A linear systems approach to flow control. *Ann. Rev. Fluid Mech.*, 39:383–417, 2007.

C. Lee, J. Kim, D. Babcock, and R. Goodman. Application of neural networks to turbulence control for drag reduction. *Physics of Fluids*, 9(6):1740–1747, 1997.

H. Li, J. Tan, Z. Gao, and B. R. Noack. Machine learning open-loop control of a mixing layer. *Phys. Fluids*, 32:111701:1–7, 2020. ISSN 1070-6631.

R. Li, D. Barros, J. Borée, O. Cadot, B. R. Noack, and L. Cordier. Feedback control of bi-modal wake dynamics. *Exp. Fluids*, 57:1–6, 2016.

R. Li, B. R. Noack, L. Cordier, J. Borée, and F. Harambat. Drag reduction of a car model by linear genetic programming control. *Exp. Fluids*, 58:103:1–20., 2017.

R. Li, B. R. Noack, L. Cordier, J. Borée, E. Kaiser, and F. Harambat. Linear genetic programming control for strongly nonlinear dynamics with frequency crosstalk. *Arch. Mech.*, 70(6):505–534, 2018.

Y. Li, W. Cui, Q. Jia, Q. Li, Z. Yang, M. Morzyński, and B. R. Noack. Explorative gradient method for active drag reduction of the fluidic pinball and slanted ahmed body. *J. Fluid Mech.*, 932:A7:1–48, 2022.

D. M. Luchtenburg, B. Günter, B. R. Noack, R. King, and G. Tadmor. A generalized mean-field model of the natural and actuated flows around a high-lift configuration. *J. Fluid Mech.*, 623:283–316, 2009.

A. Nair, C.-A. Yeh, E. Kaiser, B. R. Noack, S. L. Brunton, and K. Tiara. Cluster-based feedback control of turbulent post-stall separated flows. *J. Fluid Mech.*, 875:345–375, 2019.

J. A. Nelder and R. Mead. A simplex method for function minimization. *J. Comput.*, 7:308–313, 1965.

B. R. Noack. Closed-loop turbulence control—From human to machine learning (and retour). In Y. Zhou, M. Kimura, G. Peng, A. D. Lucey, and L. Hung, editors, *Fluid-Structure-Sound Interactions and Control. Proceedings of the 4th Symposium on Fluid-Structure-Sound Interactions and Control*, pages 23–32. Springer, 2019.

B. R. Noack, K. Afanasiev, M. Morzyński, G. Tadmor, and F. Thiele. A hierarchy of low-dimensional models for the transient and post-transient cylinder wake. *J. Fluid Mech.*, 497:335–363, 2003.

B. R. Noack, W. Stankiewicz, M. Morzyński, and P. J. Schmid. Recursive dynamic mode decomposition of transient and post-transient wake flows. *J. Fluid Mech.*, 809:843–872, 2016.

V. Parezanović, J. C. Laurentie, C. Fourment, L. Cordier, B. R. Noack, and T. Shaqarin. Modification of global properties of a mixing layer by open/closed loop actuation. In *Proceedings of the 8th International Symposium On Turbulent and Shear Flow Phenomena*, 2013.

V. Parezanović, L. Cordier, A. Spohn, T. Duriez, B. R. Noack, J.-P. Bonnet, M. Segond, M. Abel, and S. L. Brunton. Frequency selection by feedback control in a turbulent shear flow. *J. Fluid Mech.*, 797: 247–283, 2016.

M. Pastoor, L. Henning, B. R. Noack, R. King, and G. Tadmor. Feedback shear layer control for bluff body drag reduction. *J. Fluid Mech.*, 608:161–196, 2008.

B. Protas. Linear feedback stabilization of laminar vortex shedding based on a point vortex model. *Phys. Fluids*, 16(12):4473–4488, 2004.

J. Rabault, M. Kuchta, A. Jensen, U. Réglade, and N. Cerardi. Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control. *J. Fluid Mech.*, 865: 281–302, 2019.

C. Raibaudo, P. Zhong, R. J. Martinuzzi, and B. R. Noack. Closed-loop control of a triangular bluff body using rotating cylinders. In *The 20th World Congress of the International Federation of Automatic Control (IFAC)*, pages 1–6, Toulouse, France, 2017.

C. Raibaudo, P. Zhong, B. R. Noack, and R. J. Martinuzzi. Machine learning strategies applied to the control of a fluidic pinball. *Phys. Fluids*, 32:015108:1–13, 2020. ISSN 1070-6631.

K. J. Åström and R. M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, Princeton, 2010.

F. Ren, C. Wang, and H. Tang. Active control of vortex-induced vibration of a circular cylinder using machine learning. *Phys. Fluids*, 31(9):093601, 2019.

C. W. Rowley and D. R. Williams. Dynamics and control of high-Reynolds number flows over open cavities. *Ann. Rev. Fluid Mech.*, 38:251–276, 2006.

J. Seifert. A review of the Magnus effect in aeronautics. *Progress in Aerospace Sciences*, 55:17–45, 2012.

T. Shaqarin and B. R. Noack. A fast converging particle swarm optimization through targeted, position-mutated, elitism (pso-tpme. Technical Report 2207.00900 [cs.NE], arXiv, 2022.

D. Sipp, O. Marquet, P. Meliga, and A. Barbagallo. Dynamics and control of global instabilities in open-flows: a linearized approach. *Appl. Rev. Mech.*, 63:251–276, 2010.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT press Cambridge, 1998.

H. Tang, J. Rabault, A. Kuhnle, Y. Wang, and T. Wang. Robust active flow control over a range of Reynolds numbers using an artificial neural network trained through deep reinforcement learning. *Phys. Fluids*, 32:053605, 2020.

B. Thiria, S. Goujon-Durand, and J. E. Wesfreid. The wake of a cylinder performing rotary oscillations. *J. Fluid Mech.*, 560:123–147, 2006.

M. Wahde. *Biologically Inspired Optimization Methods: An Introduction*. WIT Press, 2008.

Q. L. Wang, L. Yan, G. Hu, C. Li, Y. Xiao, H. Xiong, J. Rabault, and B. R. Noack. DRLinFluids – An open-source python platform of coupling deep reinforcement learning and OpenFOAM. *Phys. Fluids*, in print, 2022.

C. J. Wood. The effect of base bleed on a periodic wake. *Journal of the Royal Aeronautical Society*, 68 (643):477–482, 1964.

Z. Wu, D. Fan, Y. Zhou, R. Li, and B. R. Noack. Jet mixing enhancement using machine learning control. *Exp. Fluids*, 59:131:1–17, 2018.

Y. Zhou, Fan D., B. Zhang, , R. Li, and B. R. Noack. Artificial intelligence control of a turbulent jet. *J. Fluid Mech.*, 897:A27:1–46, 2020.

Authors

Guy Y. Cornejo Maceda is a postdoctoral researcher at Harbin Institute of Technology at Shenzhen, working on turbulence control with machine learning methods for a fast learning of robust, feedback control laws and their visualization. He obtained his Ph.D. in 2021 at Paris-Saclay University (France) under the supervision of Profs. Bernd R. Noack and François Lusseyran with whom he accelerated the learning of feedback control laws by a factor 10 compared to previously employed machine learning control. His methodology has been demonstrated on numerical and experimental MIMO plants. Examples include drag reduction and stabilization of the fluidic pinball, stabilization of the open cavity flow, lift increase and drag reduction of a high-Reynolds number airfoil, drag reduction and side-force mitigation of a yawed truck model and even the challenging feedback smart-skin separation control over a smooth ramp with 30 actuators and 56 sensors.

François Lusseyran is currently Emeritus Research Director at the CNRS in a laboratory of Paris-Saclay University, LISN (France). His career as a fluid mechanics experimentalist has always followed the thread of the expression of nonlinearities in the space-time dynamics of various flow contexts. This began with the creation of vortex circulation at the Max Planck Institut für Strömungsforschung in 1978 in Göttingen (Germany) and now contributes to the control of vortex structures, developing in wakes and resonant flows, by searching for control laws with the help of machine learning algorithms. Other research topics have included, for example, the dynamic relationship between the heart and its arterial system (LMFE Orsay-Paris, France), transitions between gas-liquid two-phase flow regimes (LEMTA, Nancy, France, University of Illinois – Urbana-Champagne, USA), model reduction in turbine mixers and in jets with transverse flow (LEMTA, Helixor, Rosenfeld, Germany) dynamics of open cavities in incompressible regime and moderate Reynolds numbers (LISN ex-LIMSI – LFD, FIUBA-UBA, Buenos Aires, Argentina), many years of collaboration on metrological issues and data analysis (IPPT PAN, Warsaw, Poland) and reduction of temporal dynamics forced by the so-called coherent structures (CORIA, Rouen, France – LFD – LadHyX, Palaiseau, France).

Bernd R. Noack is National Talent Professor at the Harbin Institute of Technology, Shenzhen, and Honorary Professor and Chair of Turbulence Control at TU Berlin. Until 2020, he was Research Director CNRS at LIMSI, Paris-Saclay and Professor at TU Braunschweig and TU Berlin. Past affiliations include the United Technologies Research Center, Max-Planck Society, German Aerospace Center and University of Göttingen. He develops closed-loop turbulence control solutions for cars, airplanes and transport systems in an interdisciplinary effort with leading groups in China, Europe and USA/Canada. His team is advancing the frontiers of nonlinear control-oriented reduced-order models and machine learning control, an automated learning of control laws in experiment and simulation. He has co-authored over 250 refereed publications, including 2 patents, 2 textbooks, 3 other books, 3 review articles and over 120 international journal articles. His work has been honored by numerous awards, e.g., a Fellowship of the American Physical Society, a CNRS Scientific Excellence award, a Senior ANR Chair of Excellence in France, and the von Mises Award of International Association of Applied Mathematics and Mechanics. He is awarded outstanding faculty at HIT and a highly cited researcher (top 1% in Mendeley/Stanford list).